SOMIOD

Service Oriented Middleware for Interoperability and Open Data

Luís Oliveira
DEI
Politécnico de Leiria
Leiria, Portugal
2221441@my.ipleiria.pt

Rafael Perdigão
DEI
Politécnico de Leiria
Leiria, Portugal
2221442@my.ipleiria.pt

Nuno Costa
DEI
Politécnico de Leiria
Leiria, Portugal
nuno.costa@ipleiria.pt

Afonso Fernandes
DEI
Politécnico de Leiria
Leiria, Portugal
2221437@my.ipleiria.pt

Duarte Pedrosa
DEI
Politécnico de Leiria
Leiria, Portugal
2221468@my.ipleiria.pt

Marisa Maximiano
DEI
Politécnico de Leiria
Leiria, Portugal
marisa.maximiano@my.ipleiria.pt

Abstract - Este documento apresenta a arquitetura e implementação do SOMIOD, uma solução de middleware concebida para uma gestão de dados escalável, interoperável e eficiente em sistemas distribuídos. O SOMIOD organiza recursos em estruturas hierárquicas, permitindo que utilizadores independentes interajam de forma fluida com aplicações, contentores, registos e notificações através de APIs RESTful. O middleware garante um acesso uniforme e manipulação de dados, suportando notificações em tempo real para eventos específicos. Entre as principais funcionalidades destacam-se a validação de recursos baseada em XML, operações HTTP (GET, POST, PATCH, DELETE) e o disparo dinâmico de notificações (MQTT, HTTP) aquando da criação ou eliminação de registos. Esta abordagem promove flexibilidade e adaptabilidade, tornando o SOMIOD uma escolha robusta para aplicações modernas orientadas a dados.

Keywords—middleware, API RESTful, notificações, Mosquitto, XML, gestão de dados, escalabilidade

I. Introdução

A Internet das Coisas (IoT) tem emergido como uma das áreas de pesquisa e desenvolvimento mais promissoras nas últimas décadas, trazendo consigo o potencial para resolver inúmeros desafios societais. Desde questões relacionadas com as alterações climáticas até à otimização da produção e consumo de energia, a IoT apresenta-se como um elemento crucial para alcançar uma maior sustentabilidade e eficiência. Além disso, a aplicação de soluções IoT em áreas como casas inteligentes, escritórios e cidades inteligentes tem demonstrado o impacto positivo que a integração de dados pode trazer para a sociedade, governos e empresas.

Apesar do progresso significativo, muitos sistemas IoT atuais enfrentam limitações severas devido ao uso de protocolos e serviços baseados em plataformas privadas, o que restringe a interoperabilidade e dificulta a partilha de dados. Este problema, frequentemente referido como "Silo de Coisas", impede o desenvolvimento de aplicações e serviços que possam beneficiar de um ecossistema mais aberto e interconectado.

O objetivo deste trabalho é desenvolver um middleware orientado a serviços que promova a uniformização no acesso, escrita e notificação de dados, independentemente do domínio de aplicação. A abordagem baseia-se na utilização de Web Services e na estruturação de recursos com base nos padrões abertos da Web. Este modelo pretende não apenas garantir a interoperabilidade, mas também fomentar a criação de aplicações e serviços que utilizem dados abertos de forma eficiente e escalável.

II. Arquitetura Geral do Sistema

A arquitetura do sistema foi projetada para garantir escalabilidade, interoperabilidade e uma gestão eficiente dos dados, utilizando padrões abertos e tecnologias amplamente adotadas. Este middleware organiza os recursos em uma hierarquia bem definida e oferece suporte a múltiplas aplicações, promovendo a uniformização no acesso, manipulação e notificação de dados.

Visão Geral da Arquitetura

O sistema organiza os recursos em níveis hierárquicos que permitem múltiplas aplicações interagirem de forma independente. Cada aplicação é responsável pela criação e gestão dos seus containers, registos de dados (records) e notificações (notifications).

Os dados recebidos pelo middleware (Somiod) são armazenados numa base de dados dedicada. Esta base de dados serve como repositório central, garantindo que os dados estejam disponíveis para serem acedidos quando necessário, assegurando respostas rápidas e precisas. Este modelo permite que o sistema mantenha consistência e eficiência na gestão dos dados, mesmo em cenários de elevada carga.

Comunicação e Protocolos

A arquitetura utiliza os seguintes protocolos para comunicação:

- **HTTP**: Um protocolo amplamente utilizado, que permite realizar pedidos ao servidor, bem como o recebimento de notificações.
- **MQTT**: Um protocolo leve e otimizado para notificações assíncronas, usado em sistemas que exigem comunicação em tempo real com baixa latência.

Formato de Dados

A interação com o sistema é realizada através de uma API RESTful, utilizando o formato XML para transmissão de dados. Esta abordagem garante padronização, a interoperabilidade entre diferentes aplicações e sistemas, e a legibilidade também humana.

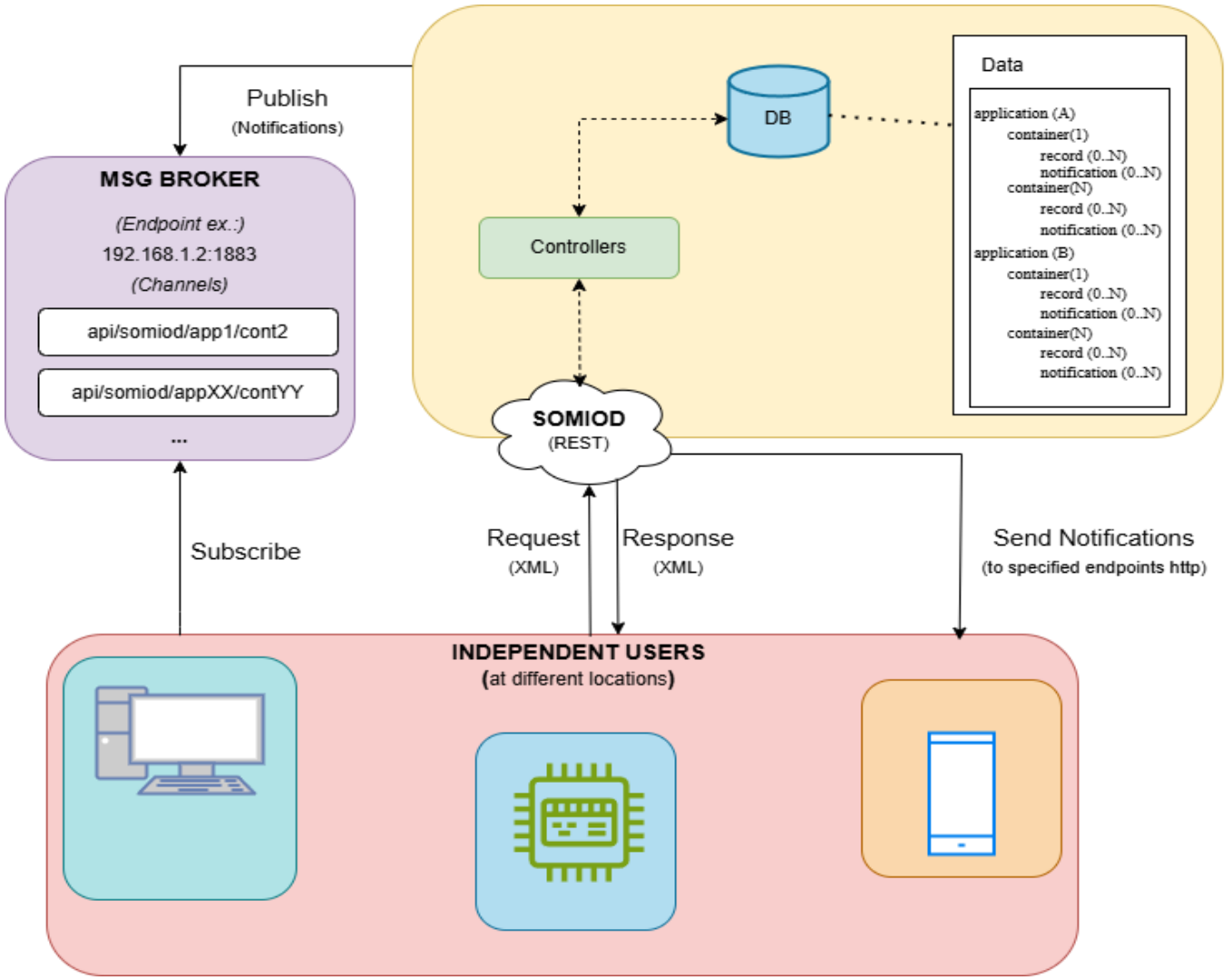

Fig.1

Arquitetura Geral

SOMIOD (REST)

O SOMIOD é o núcleo do sistema, responsável por gerir as interações entre os utilizadores e os dados armazenados na base de dados. Ele expõe uma API RESTful que permite criar, listar, localizar e eliminar os diferentes recursos organizados hierarquicamente.

- **Funcionalidades:**
  - Gerir as aplicações, containers, registos e notificações.
  - Expor endpoints para comunicação REST usando o formato de dados XML.
  - Encaminhar notificações para endpoints HTTP especificados / publicar num canal (MQTT).
- **Entrada/Saída:**
  - Entrada: Requisições (XML) de utilizadores independentes (em diferentes localizações).
  - Saída: Respostas XML com os dados solicitados, notificações, ou confirmação das operações realizadas.

Base de Dados

A base de dados é onde os dados estruturados pelo SOMIoD são armazenados. Inclui as informações sobre aplicações, containers, registos e notificações.

- **Função:**
  - Persistir os dados recebidos dos utilizadores.
  - Servir como fonte para responder a operações de consulta (GET, LOCATE).
- **Estrutura Hierárquica:**
  - Aplicação (0..N):
    - ♣ Containers (0 ..N).
      - Registos (0..N).
      - Notificações (0..N).

Controlador

Os controladores servem como a lógica de negócio do sistema, processando as operações solicitadas pelos utilizadores e encaminhando interações a base de dados.

Funções:

- Processar requisições REST, como criar, modificar, localizar recursos.
- Gerir as notificações e publicação.
- Garantir a integridade das operações na base de dados.

Recursos e Funcionalidades

Os recursos do sistema são estruturados hierarquicamente e desempenham funções específicas:

1. **Application Resource:**

   Representa uma aplicação do mundo real. O middleware suporta múltiplas aplicações simultaneamente, permitindo que cada uma crie e organize os seus próprios recursos.

2. **Container Resource:**

Funciona como uma estrutura organizacional que agrupa records e notifications associados a uma aplicação.

3. **Record Resource:**

Corresponde a um registo de dados criado dentro de um container. Os records são imutáveis após a sua criação.

4. **Notification Resource:**

Este recurso é responsável por monitorar eventos em containers, como a criação ou eliminação de registos. As notificações são disparadas automaticamente e podem ser enviadas utilizando:

a. **HTTP**: Permite que sistemas recebam notificações via chamadas de API, tanto em cenários síncronos quanto assíncronos, dependendo da implementação.
b. **MQTT**: Ideal para notificações assíncronas, com baixa latência e alta eficiência, especialmente em redes de baixa largura de banda.

TABLE I. OPERAÇÕES POR RECURSO

Recurso	***Localizar***	***Obter***	***Criar***	***Editar***	***Eliminar***
Aplicação	✅	✅	✅	✅	✅
Container	✅	✅	✅	✅	✅
Notificação	✅	✅	✅	➖	✅
Registo	✅	✅	✅	❌	✅

Users

Os utilizadores independentes representam clientes do sistema, que podem estar localizados em diferentes locais e utilizar diversos dispositivos para interagir com o SOMIOD.

- **Exemplos de Dispositivos:**
  - Computadores (PCs).
  - Dispositivos IoT (e.g., Arduino, Raspberry Pi).
  - Smartphones.
- **Interações:**
  - Requisições: Criar, modificar, localizar, eliminar, obter aplicações, containers, registos e notificações.
  - Receber Notificações: Via endpoints HTTP ou subscrições MQTT.

Msg Broker

O Message Broker é usado para gerir as notificações assíncronas através do protocolo MQTT (mosquitto). Ele publica mensagens em canais específicos, permitindo que dispositivos subscritos recebam atualizações em tempo real.

- **Funcionalidades:**
  - Publicar notificações quando eventos (criar/eliminar registos) ocorrem em containers específicos.
  - Gerir canais baseados em aplicações e containers.
- **Exemplo de Canais:**
  - api/somiod/app1/cont2
  - api/somiod/appXX/contYY

III. ARQUITETURA RESTFUL API

O SomiodController é o ponto de entrada para as requisições HTTP, centralizando a lógica de roteamento e delegação para controladores especializados. Ele implementa os métodos principais de uma API RESTful: GET (e Locate), POST, PATCH e DELETE.

A. Organização RestFul

A estrutura do endpoint segue o formato:

api/somiod/{application}/{container}/{RecNot}/{name}

Cada parte do endpoint possui um propósito específico:

- **application**: Identifica a aplicação.
- **container**: Especifica um container dentro da aplicação.
- **RecNot**: Define se a requisição é relacionada a "Record" ou "Notification".
- **name**: Identifica um recurso específico dentro de "Record" ou "Notification".

Essa estrutura garante flexibilidade, permitindo:

- Operações genéricas em aplicações ou containers.
- Operações específicas em registos ou notificações.

B. Funcionalidades Disponíveis

- **GET**

Realiza operações de leitura no sistema. O comportamento varia conforme os parâmetros enviados e os cabeçalhos HTTP:

- "somiod-locate" Header: Localiza recursos de forma hierárquica (Devolve lista de nomes).
  - Exemplo: Localizar todas as notificações de uma aplicação ou container.
- Sem o cabeçalho: Retorna detalhes de uma aplicação, container, ou recurso específico.

Na operação locate, o cabeçalho "somiod-locate: <res_type>" foi utilizado para especificar explicitamente o tipo de recurso a localizar, como application, container, record ou notification. No entanto, nas outras operações, essa abordagem não foi necessária, uma vez que o tipo de recurso pode ser determinado diretamente através do URL e/ou do conteúdo do XML request.

- **POST**

A criação de novos recursos no sistema segue uma lógica estruturada e hierárquica, com base na validação de dados XML enviados na requisição. Este processo garante consistência e conformidade com as especificações do sistema.

Lógica de Criação

- Os recursos são criados conforme a hierarquia definida. O URL da requisição aponta sempre para o recurso pai onde o novo recurso será criado.
- A estrutura XML enviada no corpo da requisição (juntamente com o URL) determina o tipo e as propriedades do recurso a ser criado.

Propriedades dos Recursos

Cada tipo de recurso possui um conjunto de propriedades específicas, conforme descrito abaixo:

application

- id: Identificador único do recurso (inteiro).
- name: Nome do recurso (string).
- creation_datetime: Data e hora de criação no formato ISO (ex.: 2024-09-25T12:34:23).

container

- id: Identificador único do recurso (inteiro).
- name: Nome do recurso (string).
- creation_datetime: Data e hora de criação no formato ISO.
- parent: Identificador único do recurso pai (inteiro).

record

- id: Identificador único do recurso (inteiro).
- name: Nome do recurso (string).
- content: Conteúdo do registo (string).
- creation_datetime: Data e hora de criação no formato ISO.
- parent: Identificador único do recurso pai (inteiro).

notification

- id: Identificador único do recurso (inteiro).
- name: Nome do recurso (string).
- creation_datetime: Data e hora de criação no formato ISO.
- parent: Identificador único do recurso pai (inteiro).
- event: Código do evento associado:
- 1 para criação.
- 2 para exclusão.
- endpoint: Endereço do endpoint associado à notificação (string). Pode ser HTTP ou MQTT, mas nunca ambos.
- enabled: Indica se a notificação está ativa ou não (valor booleano).

Regras de Implementação

- A propriedade parent armazena o identificador único do recurso pai.
- As propriedades id e name devem ser únicas em todo o sistema. Caso o nome não seja fornecido, o sistema gera automaticamente um nome único.
- O formato de data e hora segue o padrão simplificado ISO (2024-09-25T12:34:23).
- O sistema valida o XML enviado na requisição com esquemas predefinidos antes de criar os recursos.

Gatilhos de Notificação

Quando um record é criado ou excluído, um gatilho é ativado para enviar uma notificação ao endpoint associado do tipo especificado (HTTP ou MQTT).

Este modelo robusto assegura a uniformidade na criação de recursos e permite que o sistema opere de forma eficiente e escalável em ambientes distribuídos.

- **PATCH**

Atualiza recursos existentes, recebendo os novos dados no formato XML. As atualizações suportadas incluem:

- Alteração do nome de uma aplicação ou container.
- Ativação/desativação de notificações.

- **DELETE**

Remove recursos do sistema:

- Aplicações e containers podem ser removidos, eliminando seus filhos hierárquicos.
- Notificações ou registos específicos também podem ser eliminados.

C. Controladores

O SomiodController delega a lógica principal para controladores especializados:

- ApplicationController: Gerencia operações sobre aplicações.
- ContainerController: Trata containers dentro de uma aplicação.
- NotificationController: Lida com notificações associadas a containers.
- RecordController: Gerencia os registos armazenados em containers.

Exemplo de criação de um registo:

1. Um cliente realiza uma requisição POST para: api/somiod/{application}/{container}
2. O SomiodController identifica que o corpo XML contém um registo e delega a operação ao RecordController.
3. O RecordController valida o XML e insere o registo na base de dados.
4. Caso haja notificações configuradas para o container, estas são disparadas para os endpoints definidos (NotificationController).

Exemplo de localização de notificações:

1. Um cliente realiza uma requisição GET com o cabeçalho somiod-locate: notification para: api/somiod/{application}/{container}
2. O SomiodController delega a busca ao NotificationController.
3. O NotificationController retorna a lista de notificações (nomes) associadas ao container.

IV. EVALUATION

A avaliação do sistema foi realizada em dois momentos distintos: a verificação das funcionalidades internas por meio de uma aplicação de testes e a simulação de um cenário prático de uso. Os testes focaram na validação das operações básicas e comportamentos esperados do middleware, mas não incluíram testes de carga com múltiplos utilizadores.

Verificação de funcionalidades

Para garantir a conformidade e o correto funcionamento do middleware, foi desenvolvida uma aplicação de testes que permite validar todos os endpoints e operações disponíveis. Esta aplicação testou exaustivamente as funcionalidades de:

- Criação, modificação, localização e eliminação de aplicações, containers, registos e notificações.
- Configuração de notificações e disparo de eventos ao adicionar ou remover registos.

Os resultados mostraram que todas as funcionalidades foram executadas conforme o esperado, sem erros ou comportamentos inesperados.

Simulação de cenário prático

Para avaliar a aplicabilidade do sistema em cenários reais, foi simulada a utilização do middleware para controlar uma lâmpada. Esta simulação representou um cenário em que um utilizador configurava recursos no sistema para:

- Criar uma notificação associada a um container.
- Monitorizar eventos (como a adição de registos) para acionar a lâmpada em função das notificações recebidas.

A simulação demonstrou que o middleware é capaz de suportar este tipo de aplicação com sucesso. A comunicação entre a aplicação simulada e o middleware foi eficaz, com notificações enviadas e processadas corretamente.

Limitações

Embora os resultados obtidos tenham sido positivos para um único utilizador, não foram realizados testes de carga para avaliar o desempenho do sistema em cenários com múltiplos utilizadores simultâneos. Este é um aspeto que pode ser explorado em trabalhos futuros, focando na escalabilidade e no comportamento do sistema sob alta concorrência.

Conclusão

Os testes realizados evidenciam que o middleware está funcional e cumpre os objetivos propostos, tanto em termos de API quanto de capacidade de integração em cenários práticos. Apesar da ausência de testes com múltiplos utilizadores, os resultados obtidos com um único utilizador são bastante encorajadores e demonstram a viabilidade do sistema.

Conclusions and Future Work

O desenvolvimento do middleware SOMIOD resultou em uma solução funcional que atende aos objetivos propostos. O sistema demonstrou interoperabilidade ao oferecer suporte para múltiplas aplicações, organizando os recursos de forma hierárquica e padronizada. Além disso, a arquitetura foi desenhada para ser flexível e escalável, suportando cenários variados, incluindo notificações via HTTP e MQTT, adequando-se a diferentes tipos de aplicações. A usabilidade também foi um destaque, com a implementação de uma API RESTful baseada em padrões bem definidos, permitindo que desenvolvedores integrem suas aplicações de forma simples e eficaz.

Os testes realizados, tanto com a aplicação de verificação de funcionalidades quanto na simulação de um cenário prático, demonstraram que o middleware é capaz de operar de maneira confiável e atender a casos de uso reais, como o controle de dispositivos IoT. No entanto, algumas limitações foram identificadas. Não foram realizados testes de carga com múltiplos utilizadores simultâneos, o que deixa em aberto a análise do desempenho em situações de alta concorrência. Além disso, o sistema suporta notificações básicas baseadas em eventos de criação e eliminação de registos, mas não oferece suporte para lógicas de notificação mais avançadas.

Com base nos resultados obtidos e nas limitações identificadas, destacam-se algumas áreas para trabalho futuro. Primeiramente, será importante realizar testes de carga e desempenho para avaliar a escalabilidade do sistema em cenários de alta concorrência, medindo o desempenho com múltiplos utilizadores simultâneos. Em seguida, é recomendada a expansão da funcionalidade de notificações para incluir condições personalizadas e mais complexas, permitindo maior flexibilidade nas aplicações que utilizam o middleware.

Outra área relevante é a compatibilidade com outros formatos de dados, como JSON, para ampliar a usabilidade do sistema. A implementação de autenticação e controle de acesso será necessária para garantir a segurança do middleware em ambientes de produção.

O SOMIOD provou ser uma solução promissora para a gestão de recursos IoT de forma padronizada e eficiente. Apesar das áreas a serem exploradas no futuro, os resultados obtidos estabelecem uma base sólida para o desenvolvimento de sistemas IoT mais robustos, escaláveis e interoperáveis. Com a evolução do projeto, espera-se que o

SOMIOD contribua significativamente para a criação de aplicações IoT abertas e interconectadas.

V. REFERENCES

[1] Internet of Things Architecture: Recent Advances, Taxonomy, Requirements, and Open Challenges - Yaqoob, I., Hashem, I. A. T., & Ahmed, A. I. A. (2017, junho).

[2] Shaikh, N. M., Maurya, R., & Nambiar, K. A. (2021, fevereiro). Application of Restful APIs in IoT: A Review.

APPENDIX

Appendix A

Listar applicações
curl --location 'http://localhost:52431/api/somiod/' \
--header 'somiod-locate: application'

Listar containers
curl --location 'http://localhost:52431/api/somiod/' \
--header 'somiod-locate: container'

Listar notificações
curl --location 'http://localhost:52431/api/somiod/' \
--header 'somiod-locate: notification'

Listar registos
curl --location 'http://localhost:52431/api/somiod/' \
--header 'somiod-locate: record'

Aceder a uma aplicação
curl --location 'http://localhost:52431/api/somiod/App1

Aceder a um Container
curl --location 'http://localhost:52431/api/somiod/App1/Container1'

Aceder a uma Notificação
curl --location
'http://localhost:52431/api/somiod/App1/Container1/Notification/Notification2a'

Aceder a um registo
curl --location
'http://localhost:52431/api/somiod/App1/Container1/Record/Record1'

Criar aplicação
curl --location 'http://localhost:52431/api/somiod/' \
--header 'Content-Type: application/xml' \
--**data** '<Application>
<name> AppTeste </name>
</Application>'

Criar container
curl --location 'http://localhost:52431/api/somiod/-AppTeste-/' \
--header 'Content-Type: application/xml' \
--**data** '<Container>
<name>ContainerTeste</name>
</Container>'

Criar record
curl --location 'http://localhost:52431/api/somiod/-AppTeste-/ContainerTeste' \
--header 'Content-Type: application/xml' \
--**data** '<Record>
<name>RecordTeste</name>
<content>temperatura elevada</content>
</Record>'

Criar notificação
curl --location 'http://localhost:52431/api/somiod/-AppTeste-/ContainerTeste' \
--header 'Content-Type: application/xml' \
--**data** '<Notification>
<name>NotTeste</name>
<endpoint>http://test.com</endpoint>
<event>1</event>
<enabled>true</enabled>
</Notification>'

Editar aplicação
curl --location --request PATCH
'http://localhost:52431/api/somiod/-AppTeste-' \
--header 'Content-Type: application/xml' \
--**data** '
<name>AppTesteV2</name>
'

Editar container
curl --location --request PATCH
'http://localhost:52431/api/somiod/AppTesteV2/containerTeste' \
--header 'Content-Type: application/xml' \
--**data** '<name>ContainerTesteV2</name>’

Editar notificação
curl --location --request PATCH
'http://localhost:52431/api/somiod/AppTesteV2/ContainerTesteV2/Notification/NotTeste' \
--header 'Content-Type: application/xml' \
--**data** '<enabled>false</enabled>’

Eliminar notificação
curl --location --request DELETE
'http://localhost:52431/api/somiod/AppTesteV2/ContainerTesteV2/Notification/NotTeste' \
--header 'Content-Type: application/xml' \
--**data** ''

Eliminar record
curl --location --request DELETE
'http://localhost:52431/api/somiod/AppTesteV2/ContainerTesteV2/Record/RecordTeste' \

Eliminar container
curl --location --request DELETE
'http://localhost:52431/api/somiod/AppTesteV2/ContainerTesteV2'
\
--**data** ''

Eliminar aplicação
curl --location --request DELETE
'http://localhost:52431/api/somiod/AppTesteV2' --**data** ''

Appendix B

Critérios	2221441	2221468	2221442	2221437
CRUD+Locate operations for Application resource	5%	5%	5%	85%
CRUD+Locate operations for Container resource	5%	5%	85%	5%
CRUD+Locate operations for Record resource	5%	85%	5%	5%
CRUD+Locate operations for Notification resource	85%	5%	5%	5%
Testing applications	25%	25%	25%	25%
Project report	25%	25%	25%	25%

Notas:
A base de dados já vem preenchida com valores padrão.
É importante correr em primeiro lugar o IS_PROJECT_SOMIOD (se pretender usufruir do mosquitto este deverá estar a correr antes de iniciar o SOMIOD), após estes passos poderá utilizar a aplicação de testes Base_controll_Middleware, o cenário pré-criado ou poderá desenvolver o seu próprio sistema!!!