SOMIOD

Service Oriented Middleware for Interoperability and Open Data

Ana Martins
Departamento de Engenharia Informática
Instituto Politécnico de Leiria
Leiria, Portugal
2201711@my.ipleiria.pt

André Pinto
Departamento de Engenharia Informática
Instituto Politécnico de Leiria
Leiria, Portugal
2201723@my.ipleiria.pt

Ivo Bispo
Departamento de Engenharia Informática
Instituto Politécnico de Leiria
Leiria, Portugal
2200672@my.ipleiria.pt

Sara Martins
Departamento de Engenharia Informática
Instituto Politécnico de Leiria
Leiria, Portugal
2201757@my.ipleiria.pt

Marisa Maximiano
Departamento de Engenharia Informática
Instituto Politécnico de Leiria
Leiria, Portugal
marisa.maximiano@ipleiria.pt

Humberto Ferreira
Departamento de Engenharia Informática
Instituto Politécnico de Leiria
Leiria, Portugal
humberto.ferreira@ipleiria.pt

Abstract

É cada vez mais importante para as organizações serem capazes de se integrar e comunicar com uma variedade de aplicações e serviços, independentemente do seu domínio ou plataforma. No entanto, isto pode ser um desafio devido às diversas formas de acesso aos dados, escritos e notificados por diferentes sistemas.

Para abordar esta questão, concebemos e implementámos um middleware orientado para serviços que promove a interoperabilidade e a abertura de dados, normalizando a forma como os dados são acedidos, escritos, e notificados. Este middleware foi concebido para ser de domínio agnóstico, permitindo a qualquer pessoa aceder a dados e criar ou ampliar aplicações e serviços utilizando uma estrutura de recursos consistente baseada na web e serviços web baseados em normas abertas.

Neste relatório, descrevemos a concepção e implementação do middleware orientado para serviços, incluindo os serviços web e a estrutura de recursos baseada na web que adoptámos. Também discutimos os benefícios desta abordagem, incluindo a capacidade de integrar e comunicar facilmente com uma variedade de aplicações e serviços, e a capacidade de promover dados abertos e interoperabilidade. Finalmente, apresentamos alguns resultados preliminares dos nossos testes e avaliação do middleware e discutimos o trabalho futuro que poderia ser feito para melhorar ainda mais a sua funcionalidade e desempenho.

Keywords— integração, agnóstico, Web

I. Introduction

À medida que as organizações dependem cada vez mais de uma variedade de aplicações e serviços para apoiar as suas operações, a capacidade de se integrarem e comunicarem facilmente com estes sistemas está a tornar-se mais importante. No entanto, a diversidade de plataformas e formatos de dados utilizados por diferentes sistemas pode fazer com que seja um desafio alcançar a interoperabilidade.

Para abordar esta questão, empreendemos um projecto para conceber e implementar um middleware orientado para os serviços que promove a interoperabilidade e os dados abertos, normalizando a forma como os dados são acedidos, escritos, e notificados. Este middleware foi concebido para ser de domínio agnóstico, permitindo a qualquer pessoa aceder a dados e criar ou ampliar aplicações e serviços utilizando uma estrutura de recursos consistente baseada na web e serviços web baseados em normas abertas.

Neste relatório, descrevemos a concepção e implementação do middleware orientado para serviços, incluindo os serviços web e a estrutura de recursos baseada na web que adotámos. Também discutimos os benefícios desta abordagem e apresentamos alguns resultados preliminares dos nossos testes e avaliação do middleware. Finalmente, delineamos alguns trabalhos futuros que poderiam ser feitos para melhorar e expandir as capacidades do middleware.

II. System Architecture

A arquitectura do sistema para este projecto envolve a implementação de um middleware orientado para o serviço, utilizando princípios RESTful. Este middleware visa normalizar a forma como os dados são acedidos, escritos, e notificados, independentemente do domínio específico da aplicação. Ao aderir a uma estrutura uniforme, este middleware promove a interoperabilidade e os dados abertos, permitindo a qualquer pessoa aceder e utilizar dados, bem como a construção consistente de novos serviços e aplicações.

Para atingir estes objectivos, o middleware foi construído utilizando serviços Web e uma estrutura de recursos baseada em normas Web abertas. Isto permite a criação de uma arquitectura orientada para os serviços que é flexível, escalável, e facilmente integrável com outros sistemas.

Optámos por guardar a data da criação de cada recurso (*application*, *module*, *data* e *subscription*) no formato DateTime, utilizando o momento de inserção na base de dados para obter esse valor. Utilizámos esta abordagem pois

desta forma evita-se passar esta responsabilidade ao cliente, de enviar uma string com a data da criação do recurso, correndo o risco de a data enviada estar num formato errado ou de ser uma data diferente da data de criação. Assim, facilita esta validação.

Nos momentos de criação dos recursos, optámos também por não obter do body o id do novo recurso, sendo este sempre gerado automaticamente na base de dados, de forma sequencial. Assim, para aceder aos diversos recursos (excetuando o *data*), é através do nome, visto que por norma os recursos são acedidos por esse atributo (que tem a constraint unique na base de dados), não havendo assim a necessidade de o cliente usar ids.

Quanto ao processo de eliminação optámos por permitir apagar qualquer tipo de recurso, pelo que é importante ter em consideração as possíveis dependências desse recurso. Por exemplo, se tentar eliminar um recurso do tipo *module* que tenha várias *subscriptions* e *data* associados, a simples eliminação do *module* irá eliminar também esses recursos associados. Por essa razão, é importante avaliar cuidadosamente se essa ação é realmente desejada antes de prosseguir com a eliminação.

O nome do tópico de comunicação para o Mosquitto é exclusivo para cada *application*, sendo este o resultado da concatenação do nome da *application* com o nome do respetivo *module*.

Nos posts do *data*, além de guardar na base de dados realiza-se um publish para o message broker concatenando o tipo de evento “creation” com o content no formato XML, separando com ‘;’. No delete do *data*, além de apagar na base de dados realiza-se um publish para o message broker concatenando o tipo de evento “deletion” com o content vazio no formato XML, separando com ‘;’. As aplicações teste vão criar subscrições que podem ser associadas a eventos do tipo creation, deletion ou creation and deletion. Dependendo do tipo de evento a que a aplicação se subscreveu depois é feita uma validação de forma a ignorar ou não a notificação proveniente do message broker. Por exemplo, uma aplicação que subscreva com um tipo de evento “creation” vai apenas receber os publishes associados aos posts do *data*.

Em geral, a implementação de um middleware RESTful utilizando serviços Web e uma estrutura de recursos baseada na Web fornecerá um método fiável e eficiente para padronizar a forma como os dados são acedidos, escritos, e notificados, permitindo a criação e extensão de aplicações e serviços de forma consistente e previsível.

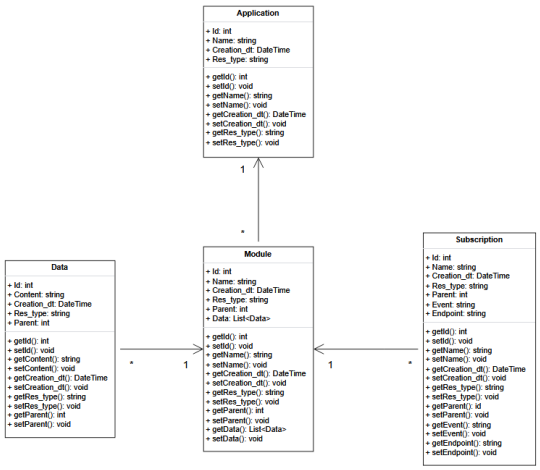


Fig. 1. Modelo Domínio

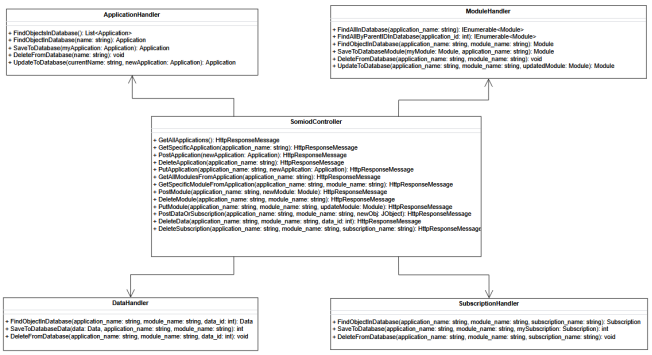


Fig. 2. Diagrama de Classes

A. *Application*

O software desenvolvido permite que múltiplas *applications* sejam suportadas e registadas de maneira consistente, cada uma com um ID único, nome e data de criação. Além disso, é possível realizar operações de criação, modificação, eliminação e listagem de todas as *applications* ou de uma *applications* específica, garantindo a integridade e organização das *applications* no sistema.

Ao criar uma aplicação, apenas é necessário enviar o seu nome e res_type que neste caso é "application".

B. *Modules*

Um *module* está associado a uma *application* específica, sendo que a mesma *application* pode ter vários *modules*. Cada *module* possui um ID único, nome, data de criação e uma chave que o vincula à *application* a que pertence. Além disso, é possível realizar operações de criação, modificação, eliminação e listagem de todos os *modules* ou de um *module* específico, garantindo a integridade e organização dos *modules* da *application* e permitindo que cada um desempenhe sua função de maneira eficiente e consistente.

Para criar um novo *module*, basta enviar o seu nome e o res_type ("module"). O URL do mesmo tem identificada a *application*, isso garantirá que o *module* seja registado e associado corretamente à *application*.

C. *Data*

O recurso *Data* é responsável por gerir e manter o registo dos dados associados a um *module* específico. Cada registo de dados possui um ID único, conteúdo, data de criação e uma chave que o vincula ao módulo a que pertence. Além disso, é possível realizar operações de criação e eliminação de *datas*, garantindo a integridade e organização dos mesmos nos *modules* de cada *application* do sistema.

Esse recurso é fundamental para garantir que os dados associados a cada *module* sejam armazenados e geridos de maneira consistente e eficiente. Ao seguir as regras e padrões estabelecidos para a criação e gestão de *datas*, é possível assegurar que as *datas* estejam sempre disponíveis e atualizadas para uso pelos *modules* da *application* .

D. *Subscription*

O recurso de *subscription* é responsável por gerir e manter o registo das *subscriptions* associadas a um *module* específico. Cada assinatura possui um ID único, nome, data de criação, uma chave que a vincula ao *module* a que pertence, o evento a ser monitorado e um endpoint para o qual serão enviadas as notificações. Além disso, é possível realizar operações de criação e eliminação de *subscriptions*, garantindo a integridade e organização das mesmas nos *modules* de cada *application* do sistema.

Esse recurso é fundamental para permitir que os *modules* da *application* recebam notificações quando determinados eventos ocorrem. Ao gerir as *subscriptions* de maneira consistente e eficiente, é possível assegurar que os *modules* recebam as notificações de maneira oportuna e precisa, o que pode ser crucial para o funcionamento correto e integrado da *application*.

E. Swagger

O Swagger é uma ferramenta que oferece diversas opções para testar e documentar a nossa API REST. Além de fornecer ferramentas para realizar testes de maneira rápida e eficiente, ele também permite gerar toda a documentação da API de forma automatizada, o que facilita o entendimento e uso da mesma por parte dos desenvolvedores.

Para tornar a visualização da API ainda mais intuitiva, o Swagger também oferece diversas ferramentas de interface, como exemplos de pedidos e respostas, descrições detalhadas dos recursos e métodos disponíveis, e outros recursos de interação. Isso permite-nos tornar a API mais acessível e fácil de usar.

III. Evaluation

A. Base para os testes

O Microsoft Visual Studio foi a ferramenta de desenvolvimento de software que foi utilizada para implementar as funcionalidades, recursos e aplicações de teste do nosso projeto.

Com o Microsoft Visual Studio, pudemos desenvolver e testar as aplicações de maneira mais rápida e eficiente, aproveitando as suas ferramentas e recursos para aumentar a produtividade e qualidade do nosso código. Além disso, ele permite-nos integrar facilmente as aplicações com outras ferramentas e serviços, ampliando ainda mais as possibilidades do nosso projeto.

B. Análise de dados

Para garantir a qualidade e integridade dos dados no nosso projeto, recorremos a diversas ferramentas e técnicas de teste e análise. Utilizamos a aplicação de teste desenvolvida por nós, a ferramenta Postman e os comandos CURL para verificar a correta implementação das funcionalidades e garantir que os dados são armazenados, acedidos e processados corretamente.

Essas ferramentas permitiram-nos realizar testes detalhados, verificando o comportamento das aplicações sob diferentes cenários e carregamentos de dados. Isso ajuda-nos a identificar e corrigir problemas e bugs de maneira mais rápida e eficiente, garantindo a qualidade e confiabilidade dos nossos sistemas.

IV. Integration/App Development

A. Aplicações LightA, LightB e RemoteLights

A aplicação de teste que desenvolvemos é composta por três projetos Windows Forms, que foram integrados na solução do projeto principal. Esses projetos correspondem a duas lâmpadas de exemplo (LightA e LightB) que se subscrevem a um *module* específico, enviando uma nova *subscription* por meio de um POST, e à ferramenta mosquitto broker no endpoint especificado. Outro projeto é um dispositivo remoto que gera e envia dados, como "On" e "Off", para os canais que estão subscritos ao *module* selecionado.

Essa aplicação de teste permite-nos verificar o funcionamento das *subscriptions* e notificações num cenário real, garantindo que os *modules* estejam a receber e a processar os dados de maneira correta e oportuna. Além disso, essa aplicação ajuda-nos a identificar e corrigir problemas e bugs, garantindo a qualidade e confiabilidade do nosso sistema de *subscriptions* e notificações.

B. Aplicação SomiodTestApplication

Para complementar os testes e garantir a qualidade do nosso projeto, desenvolvemos uma aplicação de teste Windows Forms com uma aba para cada recurso e campos necessários para realizar as operações CRUD (criação, leitura, atualização e eliminação). Essa aplicação está localizada dentro da pasta do projeto e permite-nos realizar testes detalhados, verificando o comportamento dos recursos sob diferentes cenários e carregamentos de dados.

Com esta aplicação podemos verificar se os recursos são criados, lidos, atualizados e eliminados de maneira correta, garantindo a integridade e qualidade dos dados do nosso sistema. Além disso, ela ajuda-nos a identificar e corrigir problemas e bugs de maneira mais rápida e eficiente, garantindo a confiabilidade e performance do nosso projeto.

V. Conclusions and Future Work

Em conclusão, o middleware orientado para os serviços que concebemos e implementamos tem o potencial de melhorar grandemente a interoperabilidade e promover dados abertos, normalizando a forma como os dados são acedidos, escritos, e notificados. Ao adotar uma estrutura consistente de recursos e serviços web baseados em normas abertas, criámos um sistema de domínio agnóstico que pode facilmente integrar-se e comunicar com uma variedade de aplicações e serviços.

Globalmente, os resultados dos nossos testes e avaliação foram positivos, com o middleware a demonstrar bom desempenho e fiabilidade. No entanto, ainda há espaço para melhorias e maior desenvolvimento.

VI. References

[1] https://swagger.io/docs/
[2] https://learn.microsoft.com/en-us/dotnet/csharp/

VII. Appendix

Appendix A

Application:

Criar uma application:

```
curl -X POST --header 'Content-Type: application/json' --
    header 'Accept: application/json' -d '{  \
   "res_type":"application", \
   "name":"escritorio" \
}' 'http://localhost:53204/api/somiod'
```

Obter uma application específica:

curl -X GET --header 'Accept: application/xml' 'http://localhost:53204/api/somiod/escritorio'

Obter todas as applications:

curl -X GET --header 'Accept: application/xml' 'http://localhost:53204/api/somiod'

Atualizar uma application:

curl -X PUT --header 'Content-Type: application/json' --header 'Accept: application/json' -d '{ \

"name": "sala", \

"res_type": "application" \

}' 'http://localhost:53204/api/somiod/escritorio'

Apagar uma application:

curl -X DELETE --header 'Accept: application/json' 'http://localhost:53204/api/somiod/escritorio'

Module:

Criar um module:

curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' -d '{ \

"name": "lampada", \

"res_type": "module" \

}' 'http://localhost:53204/api/somiod/escritorio'

Obter todos os modules de uma application:

curl -X GET --header 'Accept: application/xml' 'http://localhost:53204/api/somiod/sala/modules'

Obter um module específico:

curl -X GET --header 'Accept: application/xml' 'http://localhost:53204/api/somiod/sala/lampada'

Atualizar um module:

curl -X PUT --header 'Content-Type: application/json' --header 'Accept: application/json' -d '{ \

"name": "maquina", \

"res_type": "module" \

}' 'http://localhost:53204/api/somiod/sala/lampada'

Apagar um module:

curl -X DELETE --header 'Accept: application/json' 'http://localhost:53204/api/somiod/sala/lampada'

Data:

Criar um registo de data num module:

curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' -d '{ \

"res_type":"data", \

"content": "<root><content>ON</content></root>" \

}' 'http://localhost:53204/api/somiod/lighting/light-bulb'

Apagar um registo de data de um module:

curl -X DELETE --header 'Accept: application/json' 'http://localhost:53204/api/somiod/sala/maquina/data/82'

Subscription:

Criar uma subscription:

curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' -d '{ \

"name": "sub1", \

"res_type": "subscription", \

"event": "creation and deletion", \

"endpoint": "127.0.0.1" \

}' 'http://localhost:53204/api/somiod/sala/maquina'

Apagar uma subscription:

curl -X DELETE --header 'Accept: application/json' 'http://localhost:53204/api/somiod/sala/maquina/subscription/sub1'

Appendix B

Repartição do trabalho:

CRUD Application: Ivo e Ana

CRUD Module: André e Sara

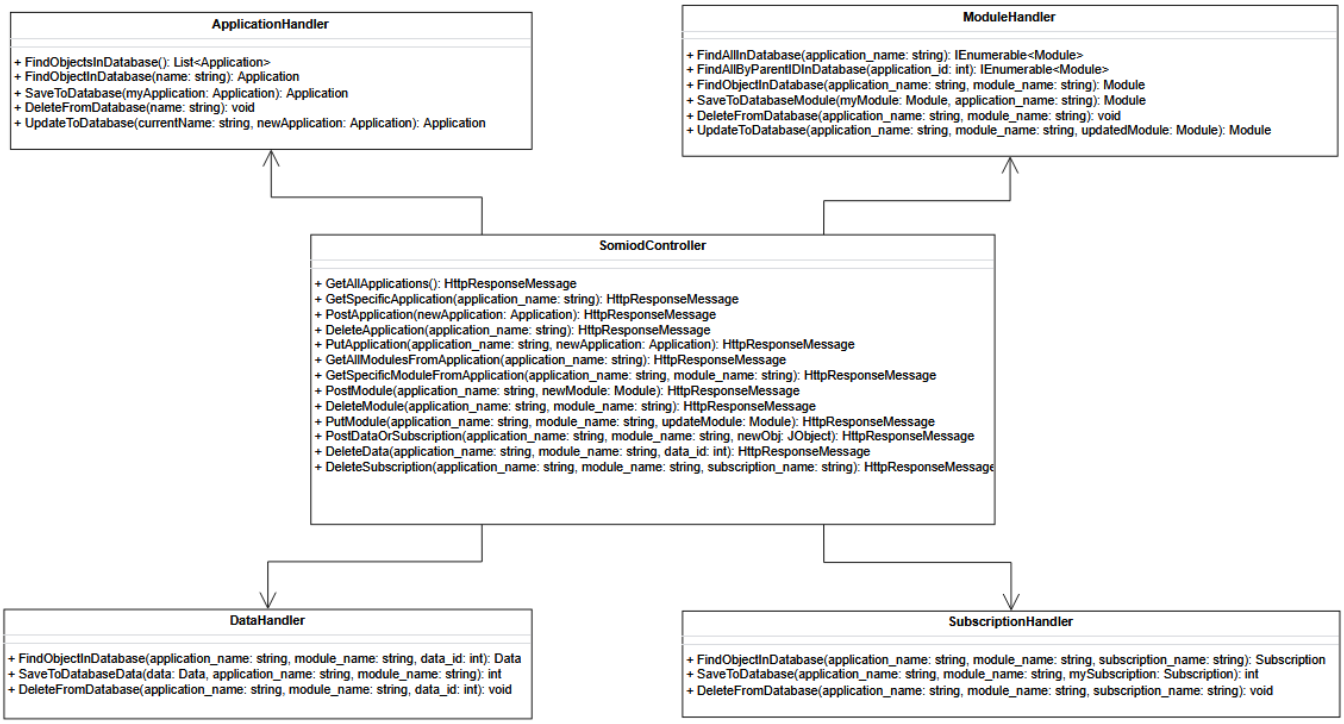
Create Data: André

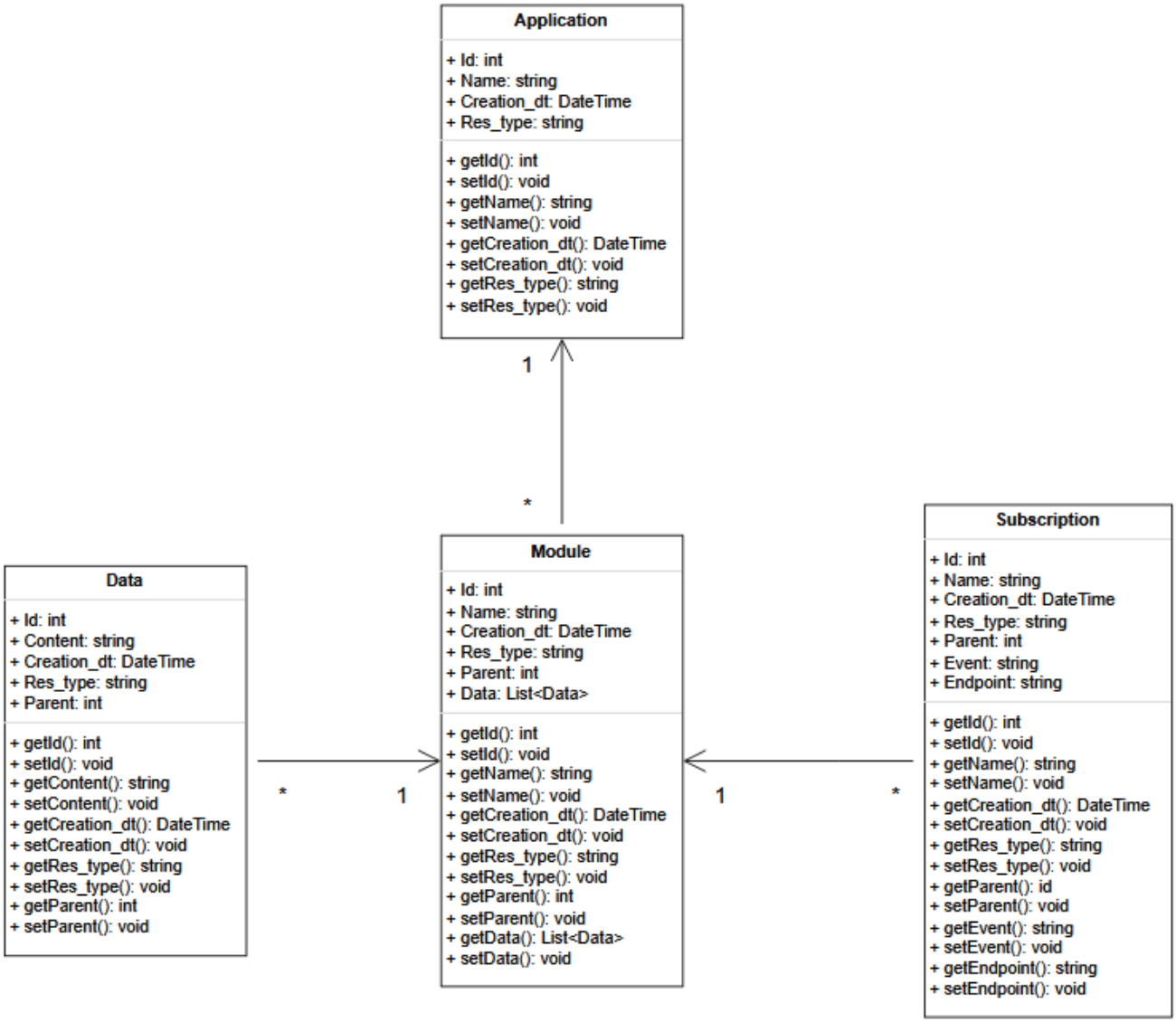
Delete Data: Sara

Create Subscription: Ana

Delete Subscription: Ivo

Aplicação de Teste: Ivo, Ana, André e Sara


ApplicationHandler
+ FindObjectsInDatabase(): List<Application>
+ FindObjectInDatabase(name: string): Application
+ SaveToDatabase(myApplication: Application): Application
+ DeleteFromDatabase(name: string): void
+ UpdateToDatabase(currentName: string, newApplication: Application): Application
ModuleHandler
+ FindAllInDatabase(application_name: string): IEnumerable<Module>
+ FindAllByParentIDInDatabase(application_id: int): IEnumerable<Module>
+ FindObjectInDatabase(application_name: string, module_name: string): Module
+ SaveToDatabaseModule(myModule: Module, application_name: string): Module
+ DeleteFromDatabase(application_name: string, module_name: string): void
+ UpdateToDatabase(application_name: string, module_name: string, updatedModule: Module): Module
SomiodController
+ GetAllApplications(): HttpResponseMessage
+ GetSpecificApplication(application_name: string): HttpResponseMessage
+ PostApplication(newApplication: Application): HttpResponseMessage
+ DeleteApplication(application_name: string): HttpResponseMessage
+ PutApplication(application_name: string, newApplication: Application): HttpResponseMessage
+ GetAllModulesFromApplication(application_name: string): HttpResponseMessage
+ GetSpecificModuleFromApplication(application_name: string, module_name: string): HttpResponseMessage
+ PostModule(application_name: string, newModule: Module): HttpResponseMessage
+ DeleteModule(application_name: string, module_name: string): HttpResponseMessage
+ PutModule(application_name: string, module_name: string, updateModule: Module): HttpResponseMessage
+ PostDataOrSubscription(application_name: string, module_name: string, newObj: JObject): HttpResponseMessage
+ DeleteData(application_name: string, module_name: string, data_id: int): HttpResponseMessage
+ DeleteSubscription(application_name: string, module_name: string, subscription_name: string): HttpResponseMessage
DataHandler
+ FindObjectInDatabase(application_name: string, module_name: string, data_id: int): Data
+ SaveToDatabaseData(data: Data, application_name: string, module_name: string): int
+ DeleteFromDatabase(application_name: string, module_name: string, data_id: int): void
SubscriptionHandler
+ FindObjectInDatabase(application_name: string, module_name: string, subscription_name: string): Subscription
+ SaveToDatabase(application_name: string, module_name: string, mySubscription: Subscription): int
+ DeleteFromDatabase(application_name: string, module_name: string, subscription_name: string): void


Application
+ Id: int
+ Name: string
+ Creation_dt: DateTime
+ Res_type: string
+ getId(): int
+ setId(): void
+ getName(): string
+ setName(): void
+ getCreation_dt(): DateTime
+ setCreation_dt(): void
+ getRes_type(): string
+ setRes_type(): void
1
*
Module
+ Id: int
+ Name: string
+ Creation_dt: DateTime
+ Res_type: string
+ Parent: int
+ Data: List<Data>
+ getId(): int
+ setId(): void
+ getName(): string
+ setName(): void
+ getCreation_dt(): DateTime
+ setCreation_dt(): void
+ getRes_type(): string
+ setRes_type(): void
+ getParent(): int
+ setParent(): void
+ getData(): List<Data>
+ setData(): void
Data
+ Id: int
+ Content: string
+ Creation_dt: DateTime
+ Res_type: string
+ Parent: int
+ getId(): int
+ setId(): void
+ getContent(): string
+ setContent(): void
+ getCreation_dt(): DateTime
+ setCreation_dt(): void
+ getRes_type(): string
+ setRes_type(): void
+ getParent(): int
+ setParent(): void
*
1
1
*
Subscription
+ Id: int
+ Name: string
+ Creation_dt: DateTime
+ Res_type: string
+ Parent: int
+ Event: string
+ Endpoint: string
+ getId(): int
+ setId(): void
+ getName(): string
+ setName(): void
+ getCreation_dt(): DateTime
+ setCreation_dt(): void
+ getRes_type(): string
+ setRes_type(): void
+ getParent(): id
+ setParent(): void
+ getEvent(): string
+ setEvent(): void
+ getEndpoint(): string
+ setEndpoint(): void