Relatório de Projeto

José Filipe Pinto Melo

Engenharia Informática

jul | 2023

GUARDA
POLI
TÉCNICO

Escola Superior de Tecnologia e Gestão

MULESOFT CUSTOM LOGGER

PROJETO EM CONTEXTO DE ESTÁGIO
PARA OBTENÇÃO DO GRAU DE LICENCIADO(A) EM
ENGENHARIA INFORMÁTICA

Professor(a) Orientador(a): Celestino Pereira Gonçalves

José Filipe Pinto Melo

Julho / 2023

Agradecimentos

É com muito orgulho que chego ao final desta etapa, uma das mais importantes da minha vida e que irá marcar o início de muitas outras. O triunfo foi maioritariamente pessoal, mas sem o apoio e força de muitos, tal não seria possível. Deste modo, esta secção é dedicada a todos os que, de um certo modo, cooperaram para que este projeto fosse possível e a todos os que fizeram desta uma fase marcante da minha vida que recordarei com saudade.

Em primeiro lugar, queria agradecer à minha namorada e família, por todo o amor, apoio e durante estes anos incentivarem-me e ensinarem que com esforço e dedicação é possível atingir os objetivos que traçamos para a nossa vida.

Em segundo lugar, queria agradecer a todos os meus amigos, por toda a ajuda, companheirismo e por todas as aventuras que vivemos nesta cidade. Foi graças a eles que esta experiência se tornou mais gratificante e mais positiva.

Um obrigado ao Professor Celestino Gonçalves por aceitar ser meu orientador de estágio e por toda a sua disponibilidade e paciência em me ajudar sempre que necessário.

Não podia deixar de agradecer ao Instituto Politécnico da Guarda e, particularmente, à Escola Superior de Tecnologia e Gestão, onde adquiri os demais conhecimentos académicos necessários para a conclusão da minha licenciatura. Agradecer ainda, ao corpo docente da licenciatura em Engenharia Informática que me acompanharam durante este processo.

Por fim, agradeço profundamente a toda a equipa da MerkleTech pela forma como me receberam e acolheram, por me ensinarem e auxiliado na execução do meu trabalho, em especial ao Miguel Pereira, supervisor responsável pelo meu estágio. A todos o meu sincero e profundo Obrigado!

Ficha de Identificação

Estudante:

Nome: José Filipe Pinto Melo

Número: 1011675

Instituição de Ensino:

Escola Superior de Tecnologia e Gestão do Instituto Politécnico da Guarda (ESTG-IPG)

Av. Dr. Francisco Sá Carneiro, nº50, 6300-559 Guarda

Telefone: +351 271 220 164 / +351 271 220 120

E-mail: estg-geral@ipg.pt

Empresa Recetora do Estágio:

MerkleTech, Unipessoal, Lda.

Edifício Adamastor Torre B, Av. Dom João II 9-1 8ª, 1990-077 Lisboa

Telefone: 211 311 216

Supervisor do Estágio Curricular na Empresa:

Miguel Sousa Pereira

Orientador do Estágio Curricular na ESTG-IPG:

Prof.º Celestino Gonçalves

Período do Estágio Curricular:

Início 14 de junho de 2021 e Fim 23 de agosto de 2021

Resumo

Este relatório descreve o trabalho desenvolvido no projeto em contexto de estágio "*Mulesoft Custom Logger*" durante o estágio na empresa MerkleTech, tendo sido este essencialmente dividido em duas fases, autoaprendizagem e aquisição de conhecimentos sobre a integração de sistemas e a tecnologia Mulesoft, e a segunda o desenvolvimento de um projeto interno da empresa.

O projeto consistiu no desenvolvimento de um componente personalizado para Mulesoft, passível de ser adicionado ao fluxo de integração, sempre que necessário registar algum dado importante para o entendimento do comportamento da aplicação. Este devera ter a capacidade de agregar esses registos em um ficheiro, que poderá ser consumido por *softwares* de interpretação e visualização de dados, facilitando e dinamizando a leitura dos mesmos.

Palavras-Chave: Mulesoft, Logger, Conector de *logger*, API, Java, Anypoint Studio, Anypoint Platform, Kibana

Abstract

This report describes the work developed in the internship project "Mulesoft Custom Logger" during the internship at MerkleTech company. It was essentially divided into two phases, self-learning and acquisition of knowledge about systems integration and Mulesoft technology, and the second one the development of an internal project for the company.

The project consisted in development of a custom component for Mulesoft, that could be added to the integration flow, whenever it is necessary to register some important data for the understanding of the application behavior. This should have the ability to aggregate these records in a file, which can be consumed by data interpretation and visualization software, making it easier and more dynamic to read them.

Key-words: Mulesoft, Logger, Conector de logger, API, Java, Anypoint Studio, Anypoint Platform, Kibana

Índice

Índice de Figuras

Lista de Siglas e Acrónimos

API	Application Programming Interface
DSL	Domain Specific Language
ESB	Enterprise Service Bus
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines
IDE	Integrated Development Environment
JDBC	Java Database Connectivity
JMS	Java Message Service
JVM	Java Virtual Machine
KQL	Kibana Query Language
REST	Transferência de Estado Representacional
SDK	Software Development Kit
SOAP	Simple Object Access Protocol
XP	Extreme Programming
DIY	Do It Yourself

1. Introdução

O presente relatório foi elaborado para descrever o projeto realizado em contexto de estágio, no âmbito da Unidade Curricular Projeto de Informática do terceiro ano da licenciatura de Engenharia Informática da Escola Superior de Tecnologia e Gestão do Instituto Politécnico da Guarda (IPG).

Este relatório fundamenta todo o trabalho desenvolvido no decorrer do estágio, iniciado no dia 14 de junho de 2021 e finalizado no dia 23 de agosto de 2021, na empresa MerkleTech.

Devido à situação pandémica atravessada na altura da realização do estágio, este teve de ser feito remotamente. No entanto, para remover as dificuldades encontradas no teletrabalho a empresa disponibilizou ferramentas como o acesso ao repositório de arquivos *Bitbucket* e a plataforma de gestão de APIs *Anypoint Platform* internos, bem como ao *Teams* da equipa de *Mulesoft* para possibilitar o acompanhamento do trabalho realizado, sem que o fator da distância fosse um problema.

Neste primeiro capítulo é dada a conhecer a entidade acolhedora, bem como o enquadramento do problema e os objetivos propostos.

1.1 Caracterização da Entidade

A empresa Merkle DACH, com sede principal na Suíça e é uma empresa que transforma negócios em serviços de clientes significativos, integrando criatividade e tecnologia. Encontra-se localizada em cinco continentes diferentes, em quarenta e cinco países e com mais de cinco mil funcionários. Esta empresa é uma agência global em expansão, tendo em Portugal quatro polos (Lisboa, Leiria, Guarda e Vila Real).

MerkleTech, é o nome da subsidiária neste país, foi constituída em 21/08/2014 tendo nove anos de atividade. A sua sede localiza-se em Lisboa. No passado a empresa já foi conhecida como Blue-Infinity, Lda., Blue-Infinity, Unipessoal Lda., e mais recentemente como TechIsobar, Unipessoal, Lda. A atividade principal da entidade

baseia-se em serviços de consultadoria de negócios e atividades nas áreas de informática de negócios, escritório, organização, gestão, informação e *marketing*, bem como comércio por grosso e a retalho de computadores e outros equipamentos informáticos. (Racios, 2022)

1.2 Motivação

Como finalista do curso de Engenharia Informática, a entrada no mercado de trabalho é algo que gera algum inquietamento, ansiedade e, acima de tudo preocupação. Desta forma, um estágio proporciona um ambiente oportuno para a aquisição de novos conhecimentos, bem como dar a oportunidade de obter o primeiro contacto no âmbito laboral e ser integrado numa equipa de trabalho.

De forma a ir ao encontro de tudo isto, visou encontrar-se uma empresa com um bom ambiente laboral, com projetos interessantes e com uma tendência de crescimento. Após a análise a várias empresas, o interesse recaiu sobre a MerkleTech, que contemplava diversas áreas e tecnologias diversificadas tais como a plataforma de integração de sistemas Mulesoft, Salesforce CRM para gestão de dados e vendas e Salesforce Commerce Cloud que é uma plataforma de *e-commerce*, nas quais seria interessante explorar e obter conhecimento.

Esta motivação foi realçada após uma partilha de opiniões com colaboradores da empresa que salientaram o potencial de desenvolvimento da área tanto ao nível de empregabilidade, progressão de carreira, bem como ao nível de aprendizagem e desenvolvimento pessoal.

1.3 Descrição do Problema

A tecnologia de integração de sistemas Mulesoft, oferece *logs*, que são registos do estado da aplicação e das suas execuções, disponibilizando ainda um componente padrão chamado "*Logger"*, com mensagens personalizáveis que pode ser incorporado no código. No entanto, estes *logs* são difíceis de estruturar e pouco legíveis, assim, é de grande interesse para a empresa que os *logs* das aplicações desenvolvidas sejam escritos de forma estruturada, correlacionada e armazenados, de forma a serem interpretados por plataformas de visualização e análise de dados.

Posto isto, é necessário o desenvolvimento de um componente personalizado para Mule 4, sendo este o motor base mais recente por trás da tecnologia Mulesoft, que seja capaz de estruturar e escrever toda a informação dos *logs* em um ficheiro com o formato de dados indicado para o efeito.

1.4 Objetivos

O desenvolvimento deste projeto pode ser dividido em duas grandes fases:

1. Aquisição de conhecimentos em integração de sistemas recorrendo a Trailheads e de Mulesoft completando o MuleSoft Development Fundamentals;
2. Aplicação dos conhecimentos adquiridos através da criação do conector de *logger* personalizado que possa ser usado em futuros projetos.

1.5 Plano de Estágio

Após a aceitação do estágio por parte da MerkleTech, Unipessoal Lda., o plano de estágio foi elaborado pelo supervisor da empresa, Miguel Pereira, em concordância com o orientador na ESTG-IPG e compreende as várias etapas a realizar no período previsto entre 14 de junho de 2021 e 10 de agosto de 2021.

O plano de atividades contou com os seguintes passos:

1. Autoaprendizagem com recurso a *Trailheads*;
2. Autoaprendizagem com recurso ao *MuleSoft Development Fundamentals*;
3. Aplicação de conhecimento em projeto interno da empresa.

1.6 Estrutura do Documento

O presente relatório, para além deste primeiro capítulo introdutório, organiza-se em mais oito capítulos. O segundo capítulo compreende o estado da arte. No terceiro capítulo é descrita a metodologia utilizada no desenvolvimento do projeto. Por sua vez, no quarto capítulo expõe a formação inicial, seguidamente, o capítulo cinco, apresenta as tecnologias utilizadas. No sexto capítulo é realizada a análise de requisitos. O sétimo capítulo tem por objetivo descrever todo o processo de implementação. No capítulo número oito são abordados os resultados dos testes realizados. E por último, o nono capítulo, as conclusões retiradas após a realização do projeto, bem como algumas considerações adicionais, como dificuldades encontradas.

2 Estado da Arte

O Estado da Arte pode ser definido como o estado atual de conhecimento ou avanço de um determinado assunto que está a ser tema de estudo. Assim, neste capítulo será apresentada uma revisão abrangente das tecnologias de integração de sistemas mais utilizadas, bem como o porquê da escolha da tecnologia Mulesoft.

2.1 Principais Alternativas

2.1.1 Apache Camel

O Apache Camel é um *framework* de integração baseado em Java. Ele oferece uma vasta gama de componentes e conectores para facilitar a integração entre diferentes sistemas e tecnologias. O Camel é conhecido por sua flexibilidade e extensibilidade, permitindo que os desenvolvedores criem rotas de integração complexas usando uma linguagem de domínio específica (DSL). Ele suporta vários protocolos, como HTTP, JDBC, FTP, JMS, entre outros. Além disso, o Camel possui um rico ecossistema de comunidade e uma documentação abrangente (Apache Software Foundation, 2004). Os seus pontos positivos contam com:

- Amplas opções de componentes e conectores para integração.
- Flexibilidade e extensibilidade.
- Comunidade ativa e documentação abrangente.

Por outro lado:

- A curva de aprendizagem pode ser íngreme para iniciantes.
- Requer conhecimento de programação Java para aproveitar todo o seu potencial.

2.1.2 IBM Integration Bus

O IBM Integration Bus é uma plataforma de integração empresarial desenvolvida pela IBM. Este fornece recursos avançados para conectar, transformar e reencaminhar mensagens entre diferentes sistemas e aplicações. A plataforma suporta uma ampla variedade de padrões de mensagens, como SOAP, REST e JMS. Este oferece recursos de transformação de dados, encaminhamento baseado em regras e orquestração de fluxo de dados, sendo também conhecido pela sua escalabilidade, desempenho e suporte a padrões de segurança (IBM, 2015). Os seus pontos a favor são:

- O suporte robusto a padrões de integração.
- Escalabilidade e desempenho.
- Conta também com recursos avançados de segurança e gestão.

Como possíveis desvantagens encontram-se:

- Pode ser uma solução mais complexa de configurar e gerir.
- Requer licenciamento e pode ter um custo mais elevado.

2.1.3 Oracle Integration Cloud

O Oracle Integration Cloud é uma plataforma de integração baseada em nuvem que simplifica a integração de aplicativos locais e em nuvem. Ele oferece recursos como mapeamento visual de dados, modelagem de processos de negócios, gestão de APIs e monitoramento em tempo real. Esta plataforma permite a criação de integrações complexas usando uma *interface* intuitiva e amigável, eliminando a necessidade de codificação manual. Ele também oferece recursos avançados de segurança e gestão de dados (Oracle, 2023). Alguns dos aspetos positivos são:

- A integração simplificada de aplicações locais e em nuvem.
- Uma interface visual intuitiva e amigável.
- Conta também com recursos avançados de gestão de APIs.

Os aspetos negativos são:

- Existem limitações em termos de personalização e extensibilidade.
- Dependência da infraestrutura em nuvem da Oracle.

2.1.4 Mulesoft

O Mulesoft é uma das plataformas de integração de sistemas mais robustas, que permite a conexão e comunicação entre sistemas diferentes e aplicações. A sua arquitetura é baseada em eventos, conhecida como *ESB (Entreprise Service Bus)*, que facilita a interoperabilidade entre sistemas. Este oferece uma ampla gama de conectores pré-construídos para interagir com sistemas externos (Munday, 2022). Alguns dos seus pontos positivos são:

- Uma interface de desenvolvimento amigável e intuitiva.
- Um ecossistema de comunidade ativa e suporte técnico robusto.
- Flexibilidade para integração em nuvem, local e híbrida.

Pode também ter algumas desvantagens como:

- O facto de o custo do licenciamento poder ser relativamente alto em comparação com outras alternativas de código aberto.
- Pode exigir investimento em formação e certificação.

2.2 Análise Crítica

Após analisar diversas alternativas de integração de sistemas, foi decidido utilizar o MuleSoft como a tecnologia a adotar para o desenvolvimento do conector personalizado de *logger*. Essa escolha se baseia no fato de que o MuleSoft ser a tecnologia utilizada pela empresa, e também por proporcionar uma integração eficiente, uma ampla comunidade de desenvolvedores e suporte de qualidade. Além disso, o MuleSoft oferece recursos avançados de conectividade, transformação de dados e monitoramento, tornando-o uma escolha ideal para atender aos requisitos do projeto.

3 Metodologia

Neste capítulo é descrita a metodologia ágil, metodologia esta usada pela empresa MerkleTech e, por conseguinte, foi a metodologia usada no desenvolvimento deste projeto.

A definição de uma metodologia no desenvolvimento de *software* é um passo fundamental para se obter um *software* de qualidade, que cumpra corretamente todos os requisitos. Entre as possíveis abordagens, optou-se pela metodologia ágil. Os métodos ágeis surgiram em 2001 e são métodos de desenvolvimento em incrementos, que se concentram no desenvolvimento rápido de *software* e lançamentos frequentes reduzindo as despesas gerais do processo, minimizando a documentação e produzindo código de alta qualidade (Consulting, 2022). O manifesto da metodologia ágil é composto por quatro valores principais:

- Os indivíduos e as suas interações são mais importantes do que processos e ferramentas;
- *Software* funcional é mais importante do que ter a documentação completa;
- A colaboração com o cliente é prioritária à negociação de contratos;
- A capacidade de responder a alterações é mais importante do que seguir um plano detalhado e linear.

Dentro das metodologias ágeis existem métodos como *Scrum*, *Lean*, *Smart* e *XP*. No caso deste projeto o método aplicado foi o *Scrum*, sendo a seguir feita uma descrição deste método e enunciada a forma como foi aplicado no desenrolar do projeto (Consulting, 2022).

Scrum é uma *framework* ágil de desenvolvimentos de produtos. Esta abordagem define uma estratégia flexível e holística de desenvolvimento de produtos. Em vez de uma abordagem sequencial, os produtos são progressivamente desenvolvidos e melhorados de forma iterativa e incremental, o que faz dele um modelo eficaz para projetos dinâmicos e suscetíveis a mudanças de requisitos, sejam eles novos ou apenas requisitos modificados (Paula, 2016).

A Figura 1 representa o funcionamento do método *Scrum*.

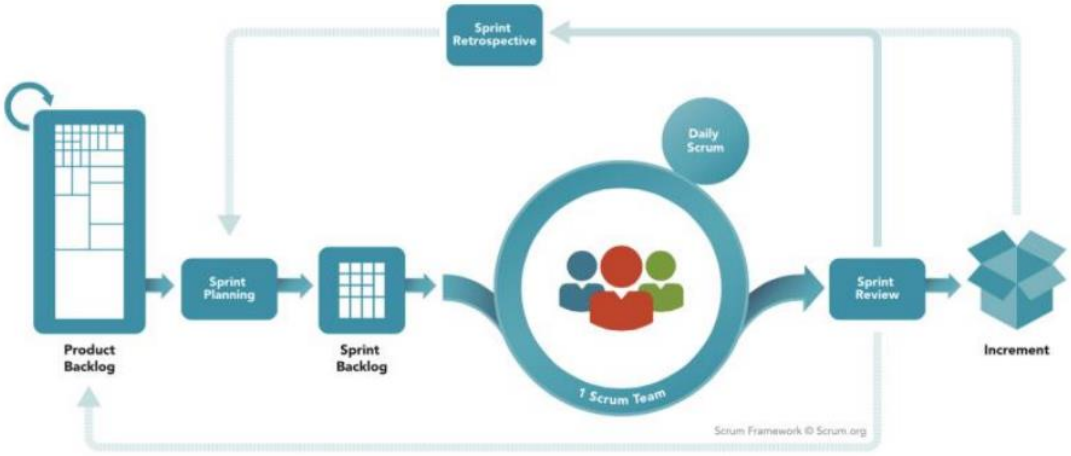

Figura 1 Funcionamento do método Scrum

Fonte: (Espiñeira, 2021)

O ciclo do *Scrum* tem o seu progresso baseado numa série de iterações, cada uma com duração de uma a quatro semanas, chamadas *Sprints*. Antes de cada *Sprint* realiza-se uma reunião de planeamento, chamada *Sprint Planning*, onde alguns requisitos do *Product Baclog* são trazidos para o *Sprint Backlog*. No decorrer da fase seguinte, a execução do *Sprint*, a equipa de desenvolvedores reúne-se diariamente (*Daily Scrum*), durante o máximo de quinze minutos, para avaliar o seu progresso. No final de cada *Sprint* é realizada uma revisão do produto do *Sprint* (*Sprint Review*) em conjunto com o cliente e *stakeholders*. Por fim, a equipa procede a uma reunião de retrospetiva do *Sprint*, que lhes permite melhorar o processo futuramente. Depois o ciclo recomeça (Paula, 2016).

No decorrer do estágio, o trabalho efetuado era apresentado ao supervisor empresarial o Miguel Pereira a cada oito dias. Após cada apresentação o trabalho era avaliado e eram discutidas quer sugestões, quer críticas construtivas para serem consideradas no próximo sprint. Para além destas avaliações, também se realizavam sessões diárias de curta duração com o supervisor e com os restantes colegas de departamento, onde se esclareciam dúvidas e se relatava o trabalho realizado no dia anterior e os pontos a atingir no próprio dia.

4 Formação Inicial

A primeira fase do estágio concentrou-se na autoaprendizagem sobre integração de sistemas recorrendo a Trailheads, explorando conceitos básicos, entendendo como funciona a integração.

Trailheads são módulos de aprendizagem interativos oferecidos pela Salesforce, que permitem aos utilizadores adquirir conhecimentos e competências específicas sobre a plataforma Salesforce por meio de exercícios, testes e tutoriais. Estes abrangem uma ampla variedade de tópicos, desde administração e desenvolvimento até vendas e *marketing*, embora, neste caso o estudo foi concentrado no tema da integração de sistemas. Esta formação teve a duração de uma semana (Salesforce Trailhead).

De seguida foram introduzidos os fundamentos do desenvolvimento com a plataforma Mulesoft, através do MuleSoft Development Fundamentals, este é um programa de aprendizagem para a plataforma Mulesoft. Ele abrange uma ampla gama de tópicos, incluindo integração de sistemas, construção de APIs, transformação de dados e gestão de erros. O programa oferece recursos educacionais, tutoriais e exercícios interativos para criar soluções de integração eficientes e robustas (Mulesoft from Salesforce).

Como pode ser visto na Figura 2 o programa é constituído por duas partes, sendo a primeira focada na aprendizagem sobre os recursos da Anypoint Platform e em como fazer *build* e *deploy* de APIs. Na segunda parte o foco recai sobre o processo de desenvolvimento de APIs, comunicação com serviços externos e recursos do *IDE* Anypoint Studio. O Mulesoft Development Fundamentals conta também com um conjunto de recursos extra como os *DIY* e um exame de preparação para a certificação que apenas fica disponível ao completar todos os módulos.

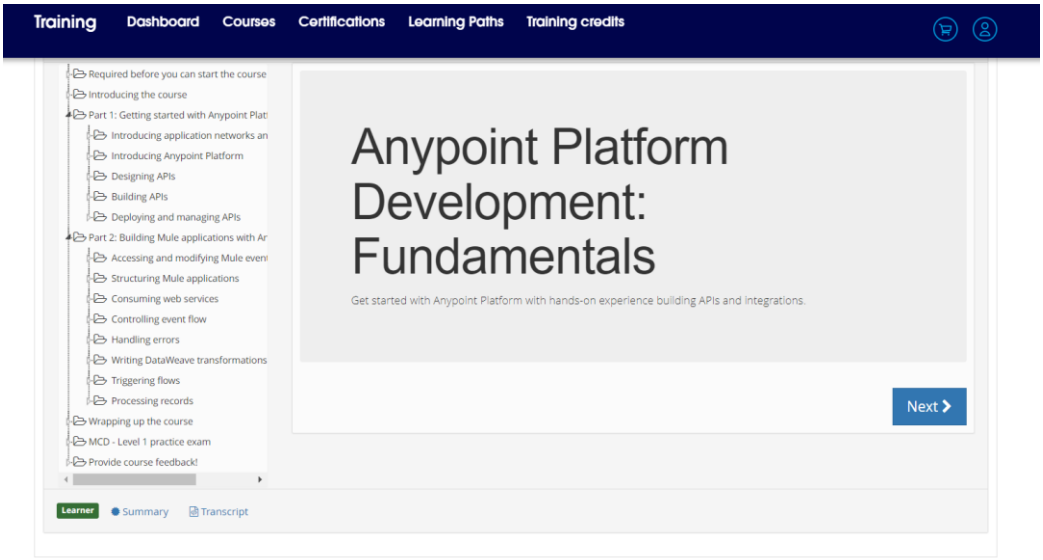

Figura 2 MuleSoft Development Fundamentals

Fonte: Elaboração própria

É também importante referir que a certificação apenas foi adquirida após o término do estágio curricular, uma vez que o tempo de duração do mesmo não era suficiente para a preparação do exame final e a conclusão do projeto proposto.

Durante essa fase, com duração aproximada de um mês, forma adquiridos conhecimentos essenciais sobre os princípios e recursos do Mulesoft, estabelecendo uma base sólida de conhecimentos sobre o funcionamento da tecnologia.

5 Tecnologias

Durante o desenvolvimento deste projeto foi necessário recorrer ao uso de várias tecnologias que serão apresentadas no decorrer deste capítulo, sendo elas:

- Anypoint Platform
- Anypoint Studio
- Apache Maven
- Conector Logger da Mulesoft
- DataWeave 2.0
- Elasticsearch
- Java
- Microsoft Teams
- Mule SDKs
- OpenJDK
- Visual Studio Code

5.1 Anypoint Platform

Está secção fornece uma visão geral do Anypoint Platform. É uma plataforma de integração baseada em nuvem que permite a criação, gestão e implementação de APIs e integrações. Os seus principais componentes são:

- **Design Center**: Permite projetar, documentar e colaborar no desenvolvimento da especificação de APIs e integrações.
- **Exchange:** É uma biblioteca de ativos reutilizáveis, tais como conectores, modelos, exemplos e APIs. O Exchange permite a partilha de ativos internamente ou com a comunidade.
- **API Manager:** Permite a gestão, governação e implementação de segurança nas aplicações.
- **Runtime Manager:** É a ferramenta que permite a implementação e gestão das aplicações de Mulesoft em contexto de nuvem, também utilizado para fazer a gestão de instâncias, monitoramento de desempenho e visualização de *logs*.

- **Monitoring:** Fornece *insights* e métricas em tempo real sobre as integrações implementadas.

(Bafna, 2021)

5.2 Anypoint Studio

Anypoint Studio é uma ferramenta de desenvolvimento gráfico fornecida pela Mulesoft para a criação e implementação de integrações na plataforma Anypoint. Este oferece uma *interface* intuitiva similar a da Figura 3, baseada em arrastar e soltar, que permite aos desenvolvedores criar fluxos de integração de maneira visual e eficiente. O Anypoint Studio contem um leque variado de conectores pré-construídos, facilitando a conexão com uma ampla gama de sistemas e serviços. Além disso, fornece recursos avançados de depuração, teste e monitoramento, permitindo aos desenvolvedores validem e solucionem problemas com os seus fluxos de integração. Neste projeto foi utilizado para importar e testar o conector desenvolvido (Mulesoft).

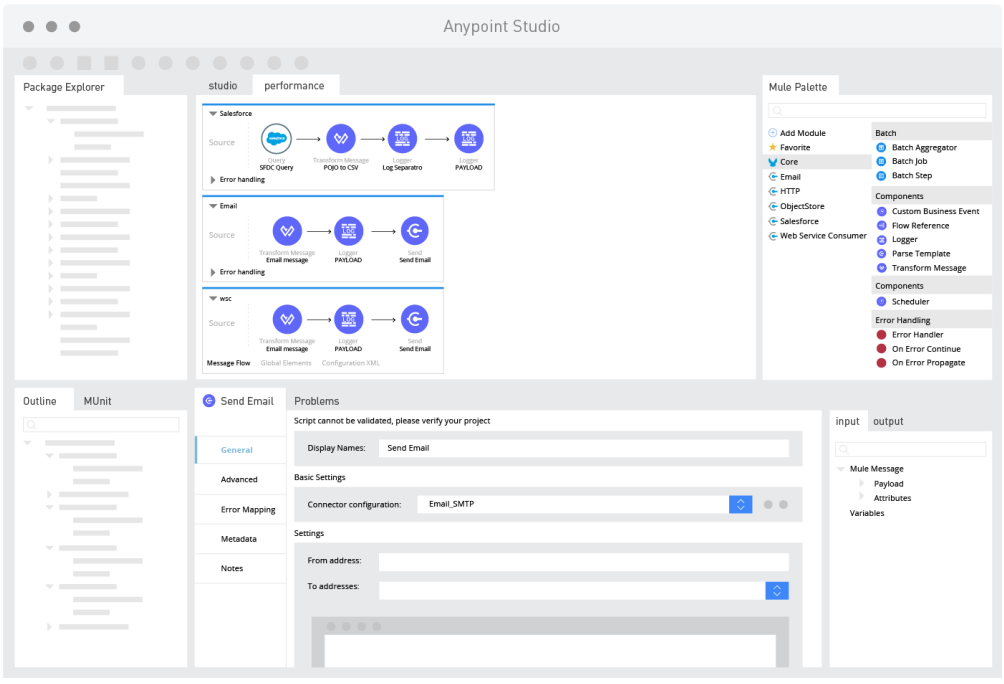

Figura 3 Interface do Anypoint Studio

Fonte: (Mulesoft)

5.3 Apache Maven

Apache Maven é uma ferramenta de gestão de projetos e automação de compilação amplamente utilizada na comunidade Java. Este fornece uma abordagem baseada em configuração e convenção para gerir dependências, compilar código-fonte, empacotar e distribuir artefactos, além de automatizar tarefas de construção comuns. O Maven utiliza um arquivo de configuração chamado pom.xml onde são definidas as informações e dependências do projeto. Com ele, os desenvolvedores podem facilmente gerir e compartilhar os seus projetos, sendo também utilizado na compilação de projetos de Mulesoft (Apache Software Foundation).

5.4 Conector Logger da Mulesoft

No contexto Mulesoft existem vários tipos de conector, estes são componentes que podem ser distribuídos ao longo do fluxo da aplicação para conectar com outros sistemas externos, manipular dados, e também, lógica ou validações.

Oferecendo funcionalidades de registo simples, mas essenciais, o conector Logger como mostrado na Figura 4, permite os desenvolvedores registarem as mensagens de *log* em diferentes níveis: *DEBUG*, *INFO*, *WARN* e *ERROR* através da configuração do parâmetro *Level*. O parâmetro *Display Name*, permite definir o nome que será exibido no fluxo, é geralmente utilizado para indicar a função do conector ou no caso do *Logger*, o que será registado. O parâmetro *Message* pode ser usado para registar mensagens contextuais como componentes da mensagem, variáveis, propriedades ou mensagens personalizadas. Por último a *Category*, esta pode ser utilizada para criar categorias para os diferentes tipos de *logs*, facilitando o rastreamento em aplicações mais complexas. Tudo isto permite que sejam rastreados e identificados problemas ou comportamentos inesperados durante a execução do fluxo de integração (Mulesoft).

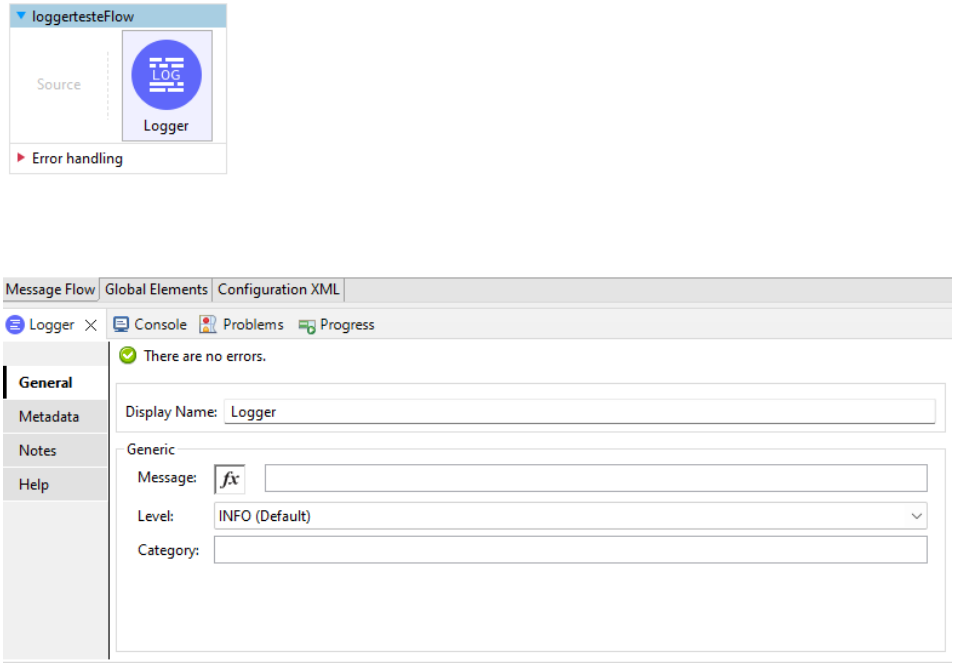

Figura 4 Conector Logger fornecido pela Mulesoft

Fonte: Elaboração própria

Os *logs* gerados pelo conector padrão são estruturados e formatados para ser apresentados no formato similar ao da Figura 5, indicando informações com o horário e data em que a mensagem foi criada, o nome da aplicação "*papi-ls-orders*", o nível de gravidade no lado esquerdo, *INFO* neste exemplo, o *event* que indica o *id* do evento, e por fim a mensagem, no exemplo "*No parcels are available for this order*".

```
10:25:31.322     07/07/2023     Worker-0     [MuleRuntime].uber.50: [papi-ls-orders-
pre].uber@org.mule.runtime.core.privileged.processor.chain.AbstractMessageProcessorChain.initialise:648 @55154a5f     INFO
event:73df35d0-1c9f-11ee-a2a4-02a66db5ac56 No parcels are available for this order.
```

Figura 5 Aparência dos logs gerados pelo conector padrão

Fonte: Elaboração própria

5.5 DataWeave 2.0

DataWeave 2.0 é uma linguagem de transformação de dados usada na plataforma Mulesoft. Esta permite a manipulação e conversão de dados em diferentes formatos, por isso sendo uma ferramenta fundamentar para a integração de sistemas. A linguagem possui uma ampla gama de funções e operadores para realizar operações como filtragem, mapeamento, agregação e junção de dados, suporte a variáveis e funções personalizadas. Sendo por tudo isso uma ferramenta essencial para desenvolvedores na plataforma Mulesoft (Erney, 2022).

5.6 Elasticsearch

O Elasticsearch é um mecanismo de busca e análise de dados distribuído e de código aberto, construído sobre o Apache Lucene. Este fornece recursos avançados para indexação, pesquisa e análise de dados em tempo real. O Elasticsearch é altamente escalável e tolerante a falhas, o que o torna adequado para lidar com grandes volumes de dados.

Uma das suas principais características é a capacidade de armazenar, indexar e pesquisar grandes quantidades de dados de maneira eficiente e rápida, permitindo executar consultas complexas e obtenha resultados relevantes de forma rápida. Além disso, o Elasticsearch suporta recursos avançados, como pesquisa em texto completo, pesquisa geoespacial, agregações de dados, auto completar e sugestões de pesquisa.

É utilizado em aplicações e sistemas que requerem recursos de busca e análise de dados em tempo real, como monitoramento de *logs*, análise de dados de negócios, pesquisa de conteúdo, recomendação de produtos, entre outros. A sua *interface* RESTful e integração com várias bibliotecas e ferramentas populares tornam-no uma escolha popular para desenvolvedores e equipes que desejam implementar recursos avançados de busca e análise em aplicações (Elasticsearch B.V).

5.7 Java

Java é uma linguagem de programação amplamente atualizada, conhecida por sua portabilidade e simplicidade. Desenvolvida pela Sun Microsystems, ela permite que os programas sejam executados em diferentes plataformas através da Java Virtual Machine (JVM). Com a sua abordagem orientada a objetos, bibliotecas padrão abrangentes e suporte da comunidade, o Java é frequentemente escolhido para o desenvolvimento de aplicativos empresariais, assim como neste caso foi utilizado como linguagem de programação para o desenvolvimento do conector (Oracle).

5.8 Microsoft Teams

O Microsoft Teams é uma plataforma unificada de comunicação e colaboração que combina *chat*, videoconferências, armazenamento de ficheiros e integração de aplicações. Este serviço integra-se no pacote do Office365 e apresenta extensões que podem ser integradas a produtos que não são da Microsoft. Todo o trabalho de colaboração e comunicação quer com o supervisor, quer com os restantes colegas foi realizado a partir desta plataforma (O'Neill, 2021).

5.9 Mule SDK

O Mule SDK (Software Development Kit) é um conjunto de ferramentas e recursos fornecidos pelo Mulesoft para o desenvolvimento de conectores personalizados para a plataforma Mule. Ele permite criar conectores que estendam as capacidades do Mule, fornecendo uma estrutura de desenvolvimento consistente, com APIs e bibliotecas que simplificam o processo de criação e configuração (Madaan, 2020).

5.10 OpenJDK

O OpenJDK (Software Development Kit) é um conjunto de ferramentas e bibliotecas fornecidas pela Oracle para desenvolver aplicações Java. Este inclui o compilador Java, a máquina virtual Java (JVM), a biblioteca padrão Java e outras ferramentas auxiliares. É um dos requisitos essenciais para o funcionamento do Anypoint Studio (Sritter, 2021).

5.11 Visual Studio Code

Visual Studio Code é um editor de código-fonte leve e altamente personalizável, desenvolvido pela Microsoft. Este fornece suporte a uma grande variedade de linguagens de programação e possui recursos avançados, como realce de sintaxe, depuração integrada, controlo de versão e extensibilidade por meio de *plugins*. Assim por está qualidade foi o escolhido como editor de código a usar no desenvolvimento do conector personalizado (Pedamkar, 2023).

6 Análise de Requisitos

Os requisitos de um sistema descrevem os procedimentos dos serviços que o sistema deve fazer, oferecer e restringir. O processo de descobrir, analisar, documentar e verificar os serviços e restrições é chamado de Engenharia de Requisitos (Machado, 2018).

Os requisitos são o ponto de partida para toda a definição de um sistema e, consequentemente, são fatores decisivos no desenvolvimento do produto final. Eles expressam as caraterísticas do *software* do ponto de vista da satisfação das necessidades do utilizador, ou seja, são caraterísticas que o *software* ou o sistema a desenvolver deverá cumprir (Machado, 2018).

Este capítulo contém a descrição dos principais requisitos identificados e documentados durante o processo de análise.

6.1 Levantamento de Requisitos

O *runtime* do Mulesoft, Mule 4, oferece *logs* do estado da aplicação e das suas execuções através do conector Logger, podendo este ser distribuído ao longo do fluxo em pontos onde se considere importante registar algum tipo de informação.

Após a análise do conector padrão, foram identificadas algumas lacunas e possíveis funcionalidades que fariam sentido adicionar a um conector personalizado de *logger*. Sendo elas:

- Necessidade de recursos avançados: O conector existente fornece funcionalidades básicas de registo, todavia seria interessante se existisse a possibilidade da adição de parâmetros extra nos *logs*, tais como, o conteúdo principal da mensagem nomeado *payload*, atributos do pedido feito a API e variáveis.
- Flexibilidade de integração: O *logger* fornecido pela Mulesoft possui integração nativa com a plataforma de gestão de APIs, a Anypoint Platform permitindo que os *logs* sejam visualizados no formato visto na Figura 2, contudo a capacidade de exportação dos *logs* em ficheiro de texto e em formato que permitisse a sua leitura

por sistemas externos ou ferramentas de análise de *logs* constituiria uma mais-valia, facilitando a centralização e gestão dos mesmos.

6.1.1 Requisitos Funcionais

O conector de *logger* personalizado deve cumprir os seguintes requisitos funcionais:

- O conector deve ser personalizável em todos os seus parâmetros de configuração: *correlation id*, *message*, *payload*, *level*, *category*, *exception details* e *variables*.
- O conector deve criar um ficheiro que contenha os *logs* gerados durante a execução da aplicação.
- O ficheiro deve ter o formato adequado para poder ser lido por *softwares* de interpretação de dados.

6.1.2 Requisitos não funcionais

O conector deve cumprir com os seguintes requisitos não funcionais:

- O conector deve ser robusto e tolerante a falhas, não prejudicando o funcionamento da aplicação.
- O conector deve ser fácil de configurar e integrar com novos projetos ou projetos existentes.

6.2 Estrutura da Mule Message

A *Mule Message* é um conceito central na plataforma Mulesoft que representa os dados e metadados transmitidos em um fluxo de integração. É uma estrutura flexível e extensível que encapsula os dados que estão a ser processados e permite a comunicação entre os componentes do fluxo. Esta é estruturada como pode ser visto na Figura 6 e contem informações essenciais como o *payload* (os dados propriamente ditos, mensagem principal), metadados (metadados adicionais) e variáveis de contexto (Mulesoft Docs).

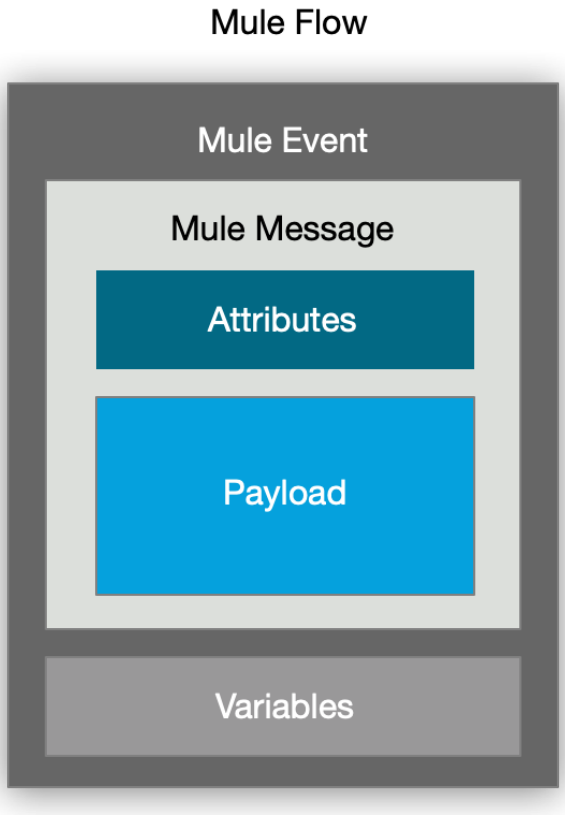

Figura 6 Estrutura da mule message

Fonte: (Mulesoft Docs)

O entendimento do seu funcionamento foi um requisito crucial para desenvolvimento do conector de *logger*, uma vez que os dados que iram ser registados pelo *logger* serão extraídos diretamente dela, utilizando a linguagem de programação própria da plataforma, Dataweave 2.0.

6.3 Classes do Mule SDK

Os capítulos anteriores foram essenciais para identificar os requisitos necessários para desenvolvimento do *logger* personalizado. Primeiramente é necessária a instalação e configuração dos pré-requisitos, o Open JDK e o Apache Maven. É aconselhado que os mesmos sejam instalados em C:\ e adicionados as variáveis de ambiente do sistema.

Para o desenvolvimento de conectores personalizados a Mulesoft disponibiliza um módulo chamado Mule SDK, este em conjunto com o Maven permite gerar as classes que constituem a base para o conector, executando o seguinte comando Maven na linha de comandos do Windows:

```
mvn org.mule.extensions:mule-extensions-archetype-maven-plugin:generate
```

Os seguintes parâmetros terão de ser definidos:

- **Nome da extensão**: Deve ser indicado o nome para o conector.
- **GroupId**: Se existir a intenção de fazer *deploy* do conector, será necessário fornecer o OrganizatioId, este pode ser consultado no Anypoint Platform.
- **Version**: A versão a atribuir ao conector, por definição é atribuído 1.0.0 – SNAPSHOT.
- **ArtifactId**: O nome do artefacto, por norma mantem se o mesmo nome da extensão.
- **Package**: Neste parâmetro deve ser passado o nome do Java *package* necessário conforme o conector que irá ser desenvolvido, neste caso será deixado a vazio e o valor padrão será assumido *org.mule.extensions.project.name.internal.*

A Figura 7 representa o processo de *build* na linha de comandos do Windows.

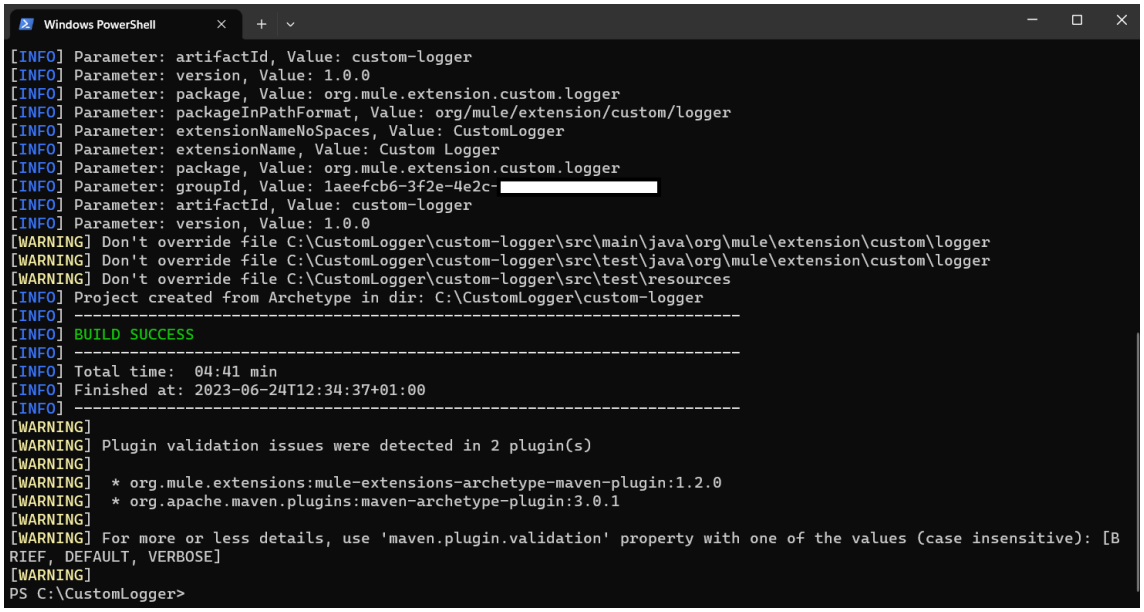

Figura 7 Geração dos ficheiros base do conector

Fonte: Elaboração própria

Ao término do processo, a estrutura base foi criada. Esta contem toda a programação essencial a construção do conector, sendo que terá de ser modificada consoante as necessidades e funcionalidades que queriam ser implementadas. As classes iniciais são: *Module*, *Configuration*, *Connection Providers*, *Components* e *Parameters*. Como pode ser visto na Figura 8.

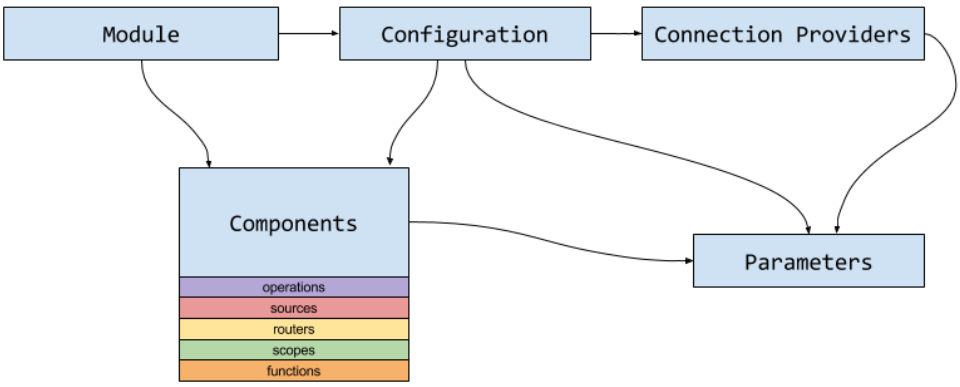

Figura 8 Estrutura básica do modulo

Fonte: (Mulesoft Docs)

6.4 Descrição e Diagrama das Classes

Nos projetos de conector baseados em Mule SDK, os nomes das classes têm o formato <**nome-do-projeto**><**nome-da-classe**>, a Figura 9 mostra a estrutura do projeto no Visual Studio Code.

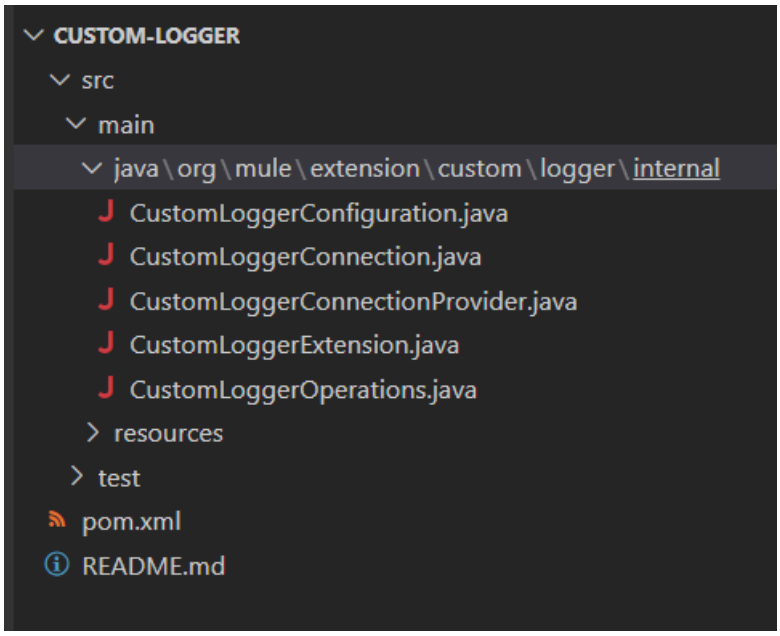

Figura 9 Estrutura do projeto

Fonte: Elaboração própria

A programação inicial das classes, foi neste momento editada para cumprir com os requisitos estabelecidos para este projeto.

Começando com a classe *CustomLoggerConfiguration*, aqui foram adicionados os seguintes parâmetros: nome da aplicação, versão e ambiente onde a aplicação será executada. Estes serão os parâmetros da configuração base do conector.

A classe *CustomLoggerConnection* é uma das mais importantes deste projeto, sendo aqui onde serão importadas as classes *org.apache.logging.log4j.LogManager* e *org.apache.logging.log4j.core.Logger*. Estas fazem parte do *framework* de *logging* Log4j2, uma biblioteca popular para registo de *logs* em aplicações Java.

```
public final class CustomLoggerConnection {

  private final Logger LOGGER = (Logger)
LogManager.getLogger("File_JSON");

  public Logger getLOGGER(){
    return LOGGER;
  }
}
```

Utilizando a classe *LogManager*, responsável por iniciar e configurar o Log4j2 no conector que está em desenvolvimento, esta fornece o método *getLogger* utilizado apara obter as mensagens de *log*.

A classe *CustomLoggerParameters* tem definidos todos os parâmetros que podem ser configurados de forma a estruturar a mensagem de *log*.

Segue se a lista de parâmetros:

- ***Correlation Id*:** Em mule 4 é um identificador único atribuído a uma mensagem ou transação específica, é usado para rastrear e correlacionar várias mensagens que passam pelo fluxo mesmo que sejam processadas por diferentes componentes ou serviços.
- ***Message*:** Este é um campo onde uma mensagem personalizada pode ser definida.
- ***Payload*:** O *payload* em Mule refere-se aos dados principais contidos em uma mensagem que está a ser processada no fluxo de integração.
- ***Level*:** O desenvolvedor pode definir o nível que a mensagem de *log* deve ter.
- ***Category*:** Permite criar categorias permitido agrupar diferentes tipos de mensagens.
- ***Exception details*:** Caso ocorra algum erro durante a execução do fluxo o desenvolvedor pode configurar este campo para exibir detalhes do erro, como por exemplo a localização de onde ocorreu ou a mensagem de erro recebida por algum dos sistemas a integrar.
- ***Variables*:** Por vezes pode ser útil saber os valores das variáveis no fluxo, este campo permite a criação de um objeto com o nome e valor de cada variável, podendo também ser configurado para imprimir apenas uma variável especifica.

Foi criada uma classe auxiliar, *LogLevel*, esta devolve os possíveis valores para o nível de *logs*. Estes podem ser *DEBUG*, *ERROR*, *INFO*, *TRACE*, *FATAL* e *WARN*.

Finalmente a classe *CustomLoggerOperations*, nesta é estruturada a mensagem de log utilizando o ObjectMessage, que permite enviar e receber objetos serializáveis como parte da comunicação assíncrona entre componentes do sistema.

Todo o código fonte de todas estas aplicações pode ser consultado no Anexo 1 – Classes do Connector.

Na Figura 10 é possível visualizar as classes anteriormente descritas, bem como as suas relações, atributos e métodos.

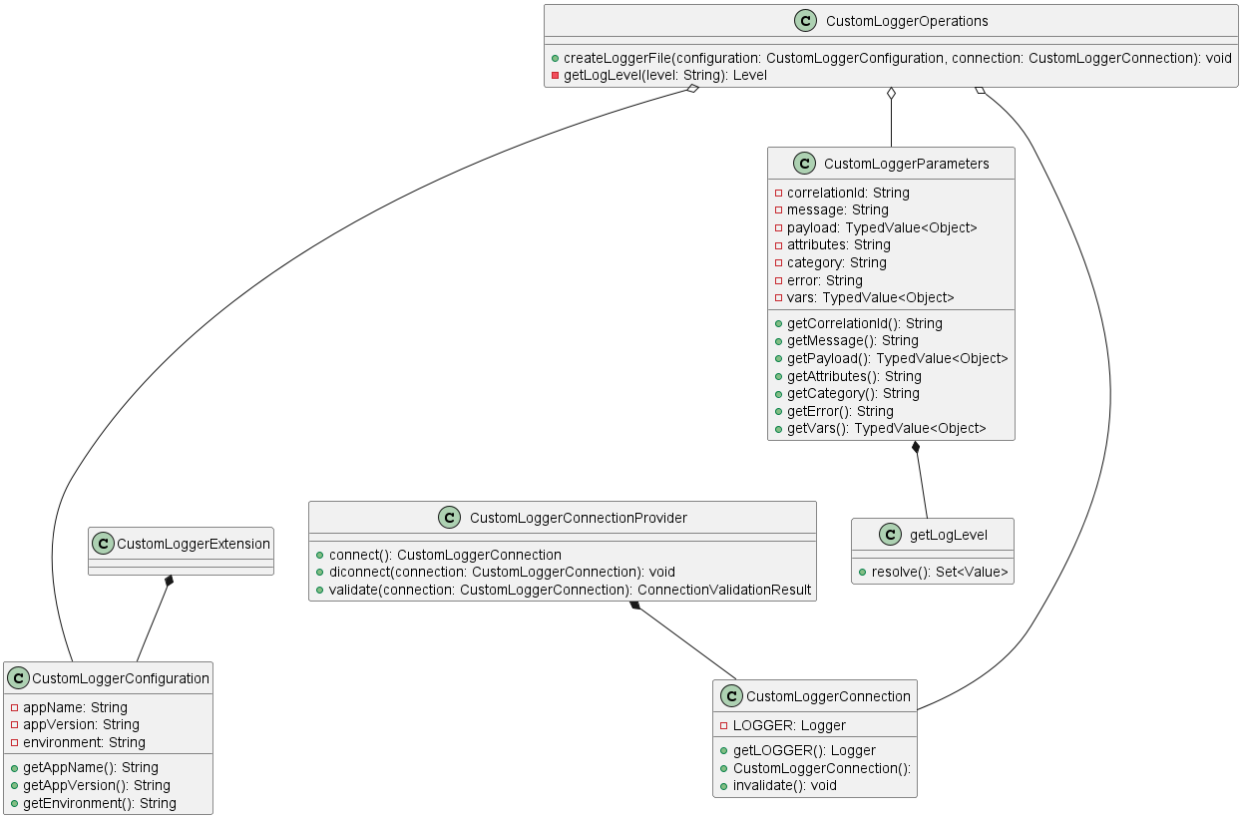

Figura 10 Diagrama de Classes

Fonte: Elaboração própria

7 Implementação

No decorrer deste capítulo, será abordado todo o processo de implementação do projeto desenvolvido durante o período de estágio.

Após o término do desenvolvimento das classes necessárias o primeiro passo na implementação consistiu no *deploy* do conector para o Anypoint Exchange, para que dessa forma este possa ser implementado em aplicações Mulesoft.

Na segunda etapa compreende a configuração do ficheiro Log4j2.xml em aplicações Mulesoft, para que o Mule permita a utilização de uma *framework* de *logger* personalizada. Apresenta também, o desenvolvimento da aplicação criada para testar o funcionamento do conector, utilizando configurações variadas.

Por fim, será abordado o *software* de visualização de dados escolhido para interpretar o ficheiro com os *logs*, Elasticsearch, mais especificamente o seu *plugin* Kibana.

7.1 Publicação do Conector

Para que seja possível a publicação do conector desenvolvido no Exchange da empresa é necessário adicionar algumas configurações ao ficheiro pom.xml, este contem todas as dependências necessárias ao projeto, ao processo de *build* e repositórios.

Este foi um processo complicado e que tomou uma parte considerável do período de estágio. Devido à dificuldade encontrada a entender a tecnologia Maven, ao grande número de incompatibilidades de *plugins* e configurações que tiveram de ser resolvidas, foram necessárias várias tentativas e alterações até que o processo de *deploy* do conector decorresse com sucesso.

A Figura 11 mostra um exemplo de um dos erros ocorridos durante o processo de *deploy* do conector.

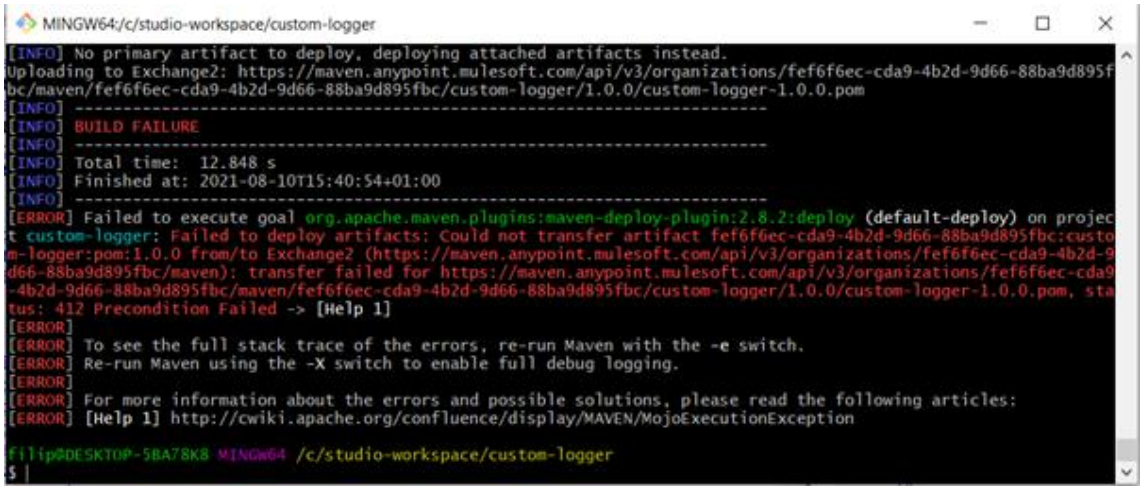

Figura 11 Erro durante o processo de deploy

Fonte: Elaboração própria

Depois de algum tempo de pesquisa em documentação oficial e em fóruns, foi possível chegar as configurações finais dos ficheiros pom.xml, presente no conector, e settings.xlm ficheiro de configuração do Maven. Estas estão indicadas no Anexo 2 - Configurações do ficheiro POM.

Por último, o comando utilizado para o processo de publicação foi:

```
mvn deploy -Pexchange -DskipTests
```

O parâmetro -*Pexchange* representa o perfil do repositório para onde será feito o envio, este deve estar indicado no ficheiro pom.xml, foi também adicionado o parâmetro -DskipTests ao comando para que o Maven não execute nenhum teste, uma vez que por definição ele executa as classes de teste presentes no projeto (Mulesoft).

A Figura 12 representa o resultado da publicação bem sucedida, utilizando as configurações presentes no Anexo 2 - Configurações do ficheiro POM. A partir deste momento o conector ficou disponível para ser utilizado em aplicações Mulesoft.

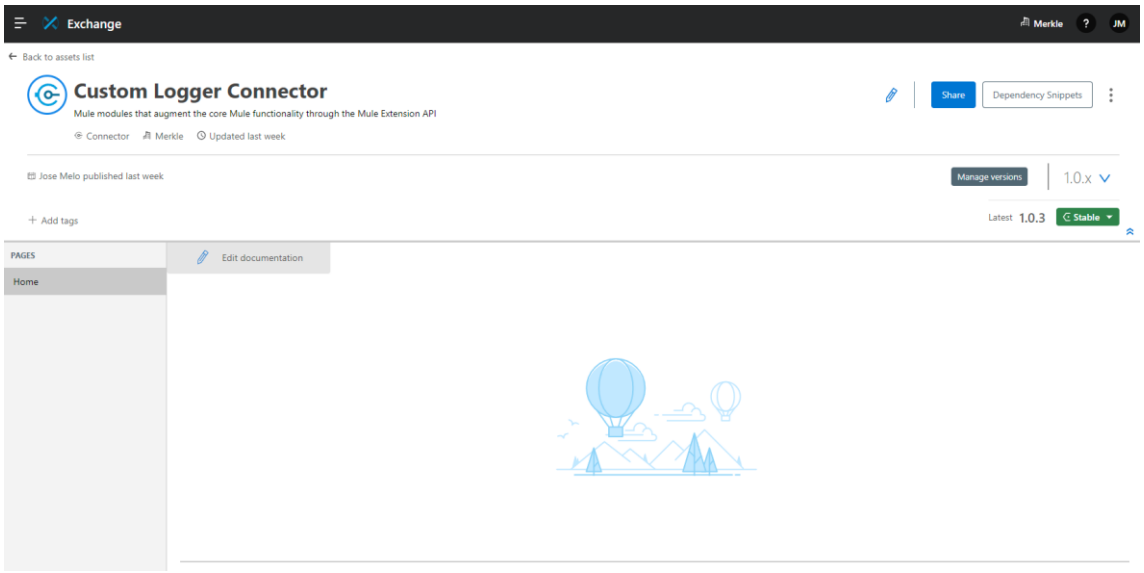

Figura 12 Resultado da publicação

Fonte: Elaboração própria

7.2 Utilização do Conector em Aplicações

Diferente dos conectores do *core* da Mulesoft, os restantes têm de ser adicionados manualmente as aplicações, para tal é necessário adicionar a *depencency* com a informação do conector ao ficheiro pom.xml da aplicação.

Na Figura 13 pode ser visto a dependência adicionada com os parâmetros:

- *groupId*: O *id* da organização que disponibiliza o conector.
- *artifactId*: Representa o nome do *asset* requerido.
- *version:* A versão do conector.
- *classifier:* O tipo do artefacto.

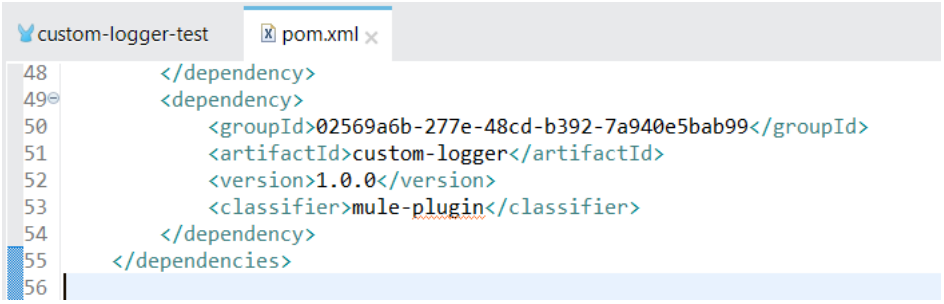

Figura 13 Dependency do custom logger connector

Fonte: Elaboração própria

O *Custom Logger* conector fica agora disponível na *Mule Palette* da aplicação, aqui é onde são exibidos todos os conectores disponíveis para ser adicionados ao fluxo. Como pode ser visto do lado esquerdo da Figura 14.

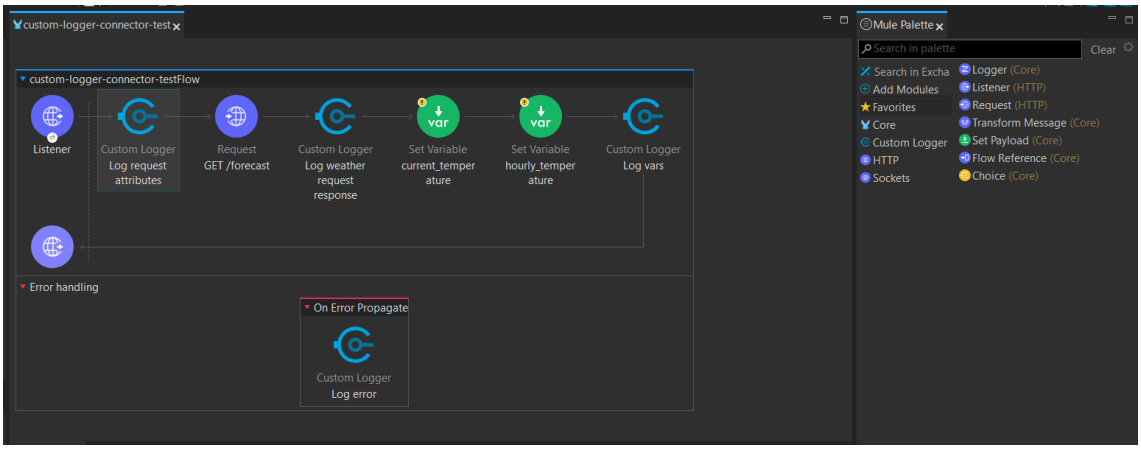

Figura 14 Distribuição dos Custom Loggers pelo fluxo

Fonte: Elaboração própria

As boas práticas de desenvolvimento sugerem que os *loggers* devem ser distribuídos pelo fluxo para fornecer a informação essencial para detetar possíveis erros e comportamentos da aplicação.

Na Figura 14 pode ser visto um exemplo de distribuição do conector *Custom Logger ao* longo do fluxo de integração, devendo ele ser disposto antes e depois de cada pedido a sistemas externos, manipulação de dados ou lógica aplicada no fluxo.

Os parâmetros de configuração do conector estão exibidos na Figura 15, estes podem ser definidos em modo literal para registar mensagens estáticas ou expressão, usando Dataweave 2.0, para configuração dinâmica dos mesmos. Note-se que em modo dinâmico as expressões são destacadas coloridamente. Pode-se alternar entre os modos clicando no botão *fx*.

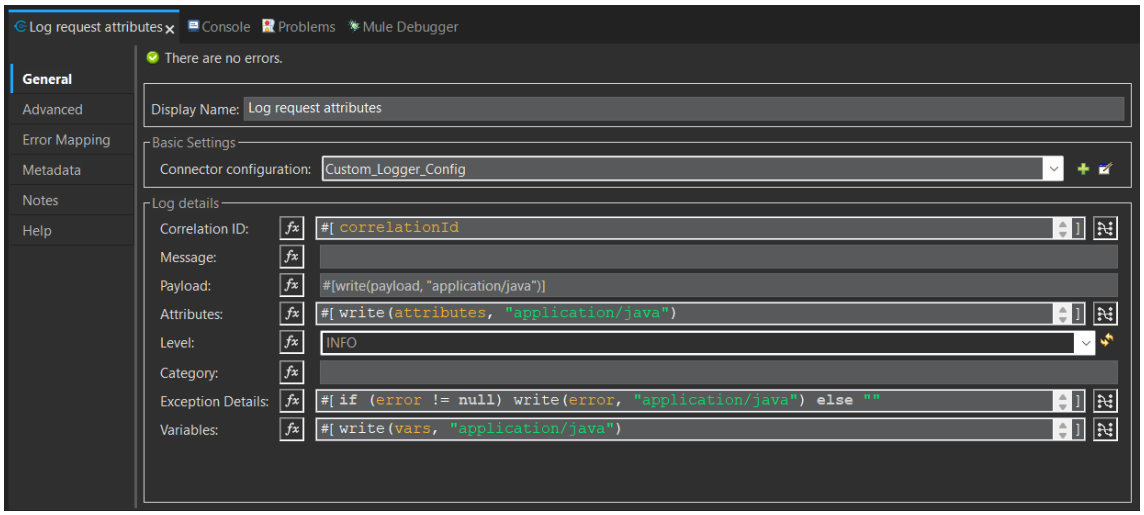

Figura 15 Interface de configuração dos parâmetros do logger

Fonte: Elaboração própria

Para que o conetor *Custom Logger* tenha acesso aos *logs* do *runtime* é necessário configurar o ficheiro log4j2.xml da aplicação, localizado em *source/main/resorces*. O script a seguir deve ser adicionado a *tag Appenders*.

```
<RollingFile name="file"
fileName="${sys:mule.home}${sys:file.separator}logs${sys:file.separato
r}mulesoft-app-template.log"
filePattern="${sys:mule.home}${sys:file.separator}logs${sys:file.separ
ator}mulesoft-app-template-%i.log">
            <PatternLayout pattern="%-5p %d [%t] [processor:
%X{processorPath}; event: %X{correlationId}] %c: %m%n"/>
            <SizeBasedTriggeringPolicy size="10 MB"/>
            <DefaultRolloverStrategy max="10"/>
 </RollingFile>
```

Neste podem ser definidos a localização, o nome do ficheiro e o tamanho máximo por arquivo, este terá um formato de saída *njson*, consumido pelo Kibana.

A configuração seguinte, deve ser adicionada na *tag* Loggers. A sua função é direcionar os eventos de *log* do apêndice JsonFile para o logger File_JSON, já configurado na classe *CustomLoggerConnection* do conector desenvolvido.

```
<AsyncLogger name="File_JSON" level="TRACE" additivity="false">
   <AppenderRef ref="JsonFile" />
</AsyncLogger>
```

Testar o funcionamento de um conector, exige o desenvolvimento de uma aplicação que conte com diferentes situações e configurações de utilização do mesmo.

O passo inicial no desenvolvimento de uma API para esta plataforma é a criação da sua especificação no Design Center da Anypoint Platform. (Mulesoft) Na especificação, são indicados os protocolos suportados pela aplicação e os *endpoints* com os seus parâmetros. Estes referem-se a URI *parameters*, usados para receber valores dinamicamente no *path* do *endpoint* e também, os *query parameters* que servem para

filtrar por um conteúdo específico, se necessário. A especificação pode também conter os *data types*, estes definem a estrutura de pedidos e respostas para cada *endpoint*.

A plataforma Mulesoft, propõe o uso de uma abordagem arquitetural denominada *API-LED Conectivity*, criando integrações de forma modular e reutilizável. Essa abordagem consiste no agrupamento das APIs por camadas sendo elas *system layer*, *process layer* e *experience layer*. Na Figura 16 podem ser observadas a *sytem layer* contendo as APIs que comunicam com sistemas externos, *a process layer* que agrupa as APIs que executam algum tipo de processamento de registos, e finalmente a *experience layer* contendo as APIs que servem de *interface* para sistemas externos ou utilizadores (Prashant Choudhary, 2023).

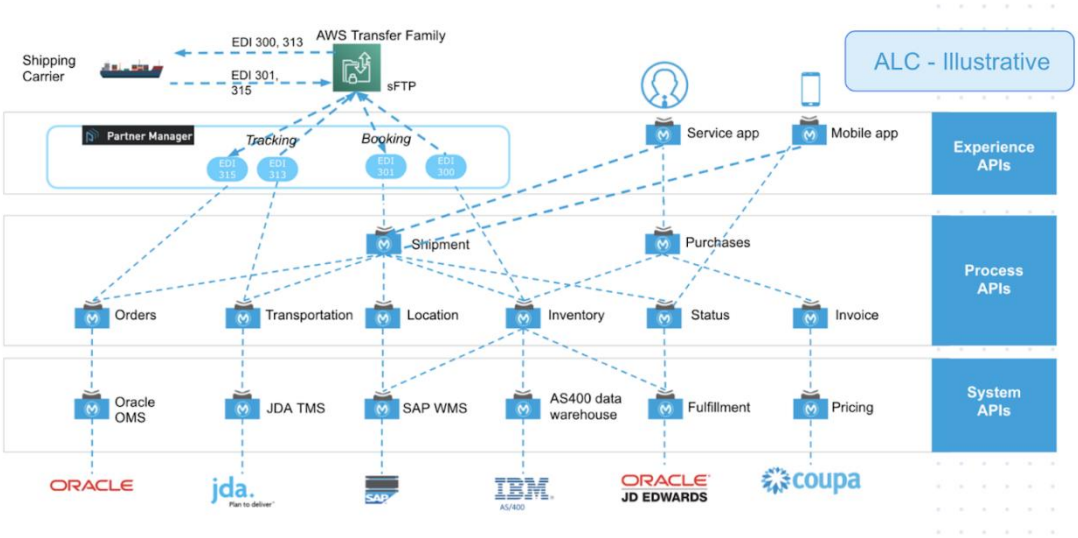

Figura 16 Exemplo de API-LED Connectivity

Fonte: (Prashant Choudhary, 2023)

A aplicação desenvolvida para o teste enquadra-se na categoria das *system* APIs, cujo objetivo é comunicar diretamente com um sistema externo, neste caso a Open Meteo API. Esta fornece dados metrológicos gratuitamente e com grande variedade parâmetros disponíveis (Open-Meteo.com).

A Figura 17 representa a especificação da API *sapi-open-meteo* no Design Center. O seu nome indica a camada a qual esta pertence seguido do nome do sistema com o qual comunica.

Figura 17 Especificação da sapi-open-meteo

Fonte: Elaboração própria

A especificação criada para a aplicação, contem apenas um *endpoint* GET /weather, este tem três *query parameters*, sendo eles latitude, longitude e *timezone*. O modelo de reposta foi definido no *data type* get-weather-response, disponível no Anexo 3 – get-weather-response.raml.

Após publicar a especificação no Exchange, o passo seguinte é criar um Mule *project* no Anypoint Studio e importar a API, a Figura 18 ilustra esse processo. É importante mencionar que será necessário fazer *login* com as credenciais de acesso a Anypoint Platform. O *endpoint* da aplicação ira ser automaticamente gerado conforme a especificação, este processo é designado por *skaffold*.

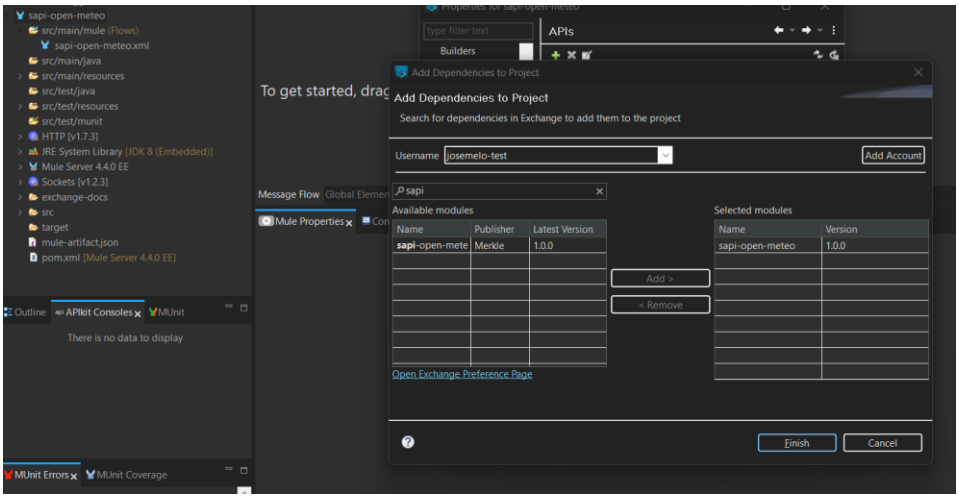

Figura 18 Importar especificação do Exchange

Fonte: Elaboração própria

O projeto da aplicação conta com a estrutura que pode ser vista na Figura 20, na *tab* do *Packege Explorer*. Seguindo esta as boas práticas de desenvolvimento e organização o ficheiro *global.xml* contem todas as configurações-base dos diferentes conectores e configurações que indicam a localização de ficheiros. Os parâmetros de configuração base dos conectores, como *host*, *port*, *path*, entre outros devem ser definidos em um ficheiro no formato *yaml*, sendo este um formato de serialização de dados, com o propósito de serem gerados ficheiros de propriedades para cada ambiente de integração.

As configurações-base do conector, como mostrado na Figura 19 contem os parâmetros *App Name*, *App Version e Environment*, estes estão definidos como propriedades no ficheiro de propriedades para o ambiente *localhost*, disponível Anexo 4 – localhost.yaml.

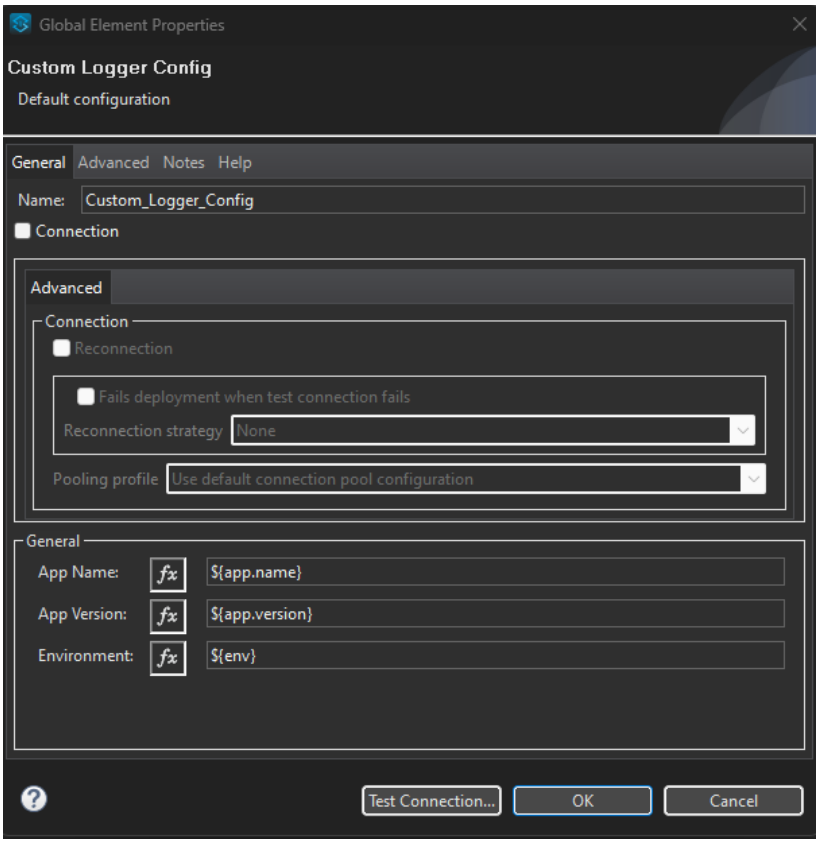

Figura 19 Configuração base do conector

Fonte: Elaboração própria

Na Figura 20 é também possível ver o fluxo de integração criado para o *endpoint* GET */weather* da aplicação.

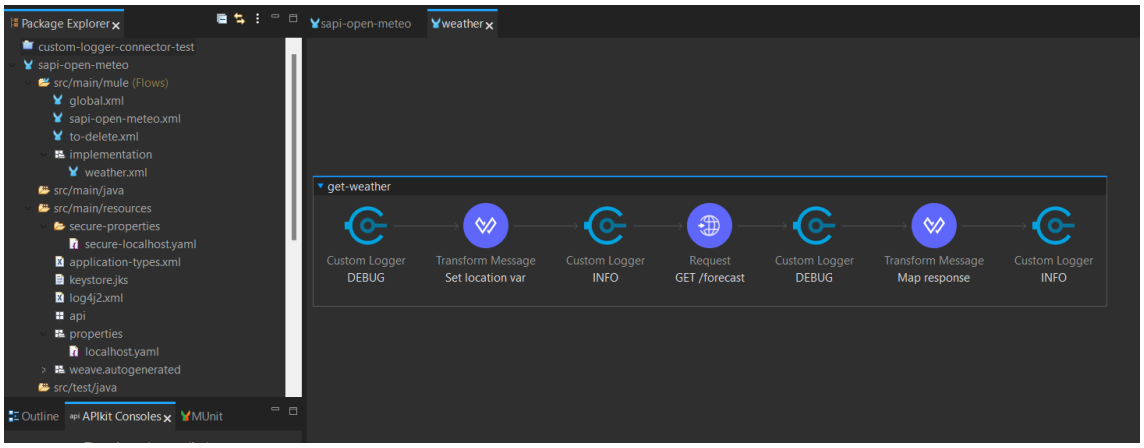

Figura 20 Fluxo para obter os dados meteorológicos

Fonte: Elaboração própria

O *logger* foi adicionado ao fluxo nos pontos onde foi considerado importante registar informação com nível *DEBUG* ou INFO. O primeiro DEBUG do fluxo representado na Figura 20 contou com a configuração mostrada na Figura 21.

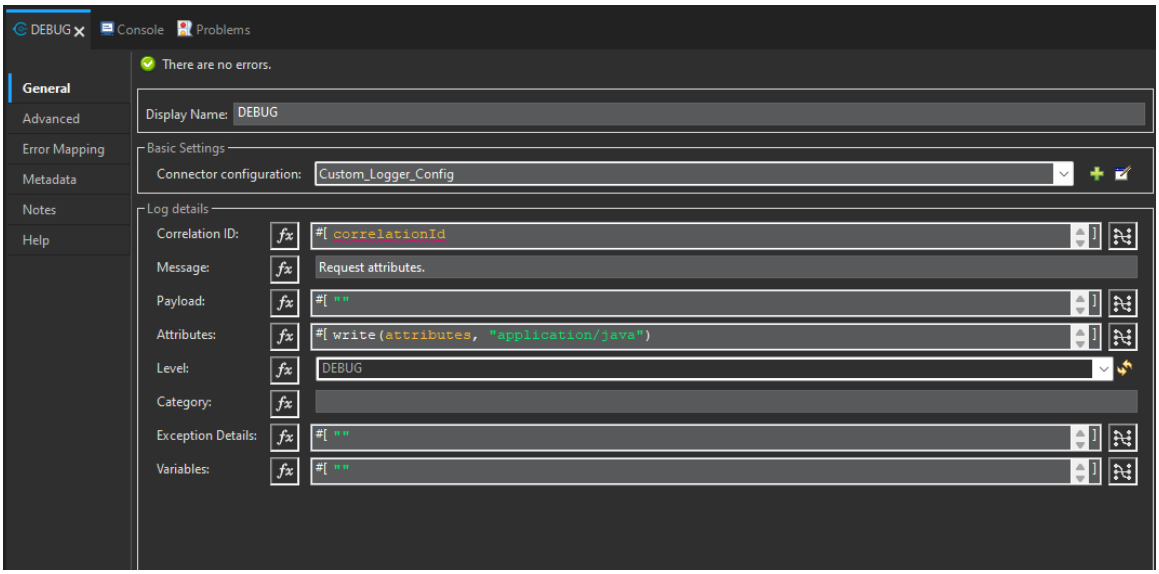

Figura 21 Exemplo 1 de configuração dos parâmetros do logger

Fonte: Elaboração própria

Na configuração mostrada é possível ver que neste caso o conector foi configurado para registar o *correlation id* (*id* da mensagem), uma mensagem estática "*Request attributes*" que indica o objetivo deste *log* e o parâmetro *Attributes* foi definido usando a

função *write* do Dataweave, passando os *attributes* do pedido recebido como parâmetro de entrada. Esta função, é usada para escrever os dados em um formato específico, neste caso a saída deverá ter um formado *application/java*. Note-se que o *Level* foi definido como *DEBUG*, uma vez que normalmente informações como atributos de pedidos, *payload* de respostas de outros sistemas e resultados de transformações de dados devem apenas ser usados para efeito de *debug* da aplicação. Os restantes parâmetros forma preenchidos com duas aspas, significando que não deve registados.

No primeiro conector de *Tansform Message*, ainda na Figura 20, é definida uma variável nomeada *location* onde são guardados os parâmetros de localização recebidos no pedido efetuado a aplicação. O conector *logger INFO* que se segue, apresenta outra possível configuração representada na Figura 22. Pode-se observar que neste exemplo apenas será registado o valor da latitude, com o intuito de apurar se será possível capturar somente o valor de um determinado campo, em casos onde o parâmetro de entrada é um objeto.

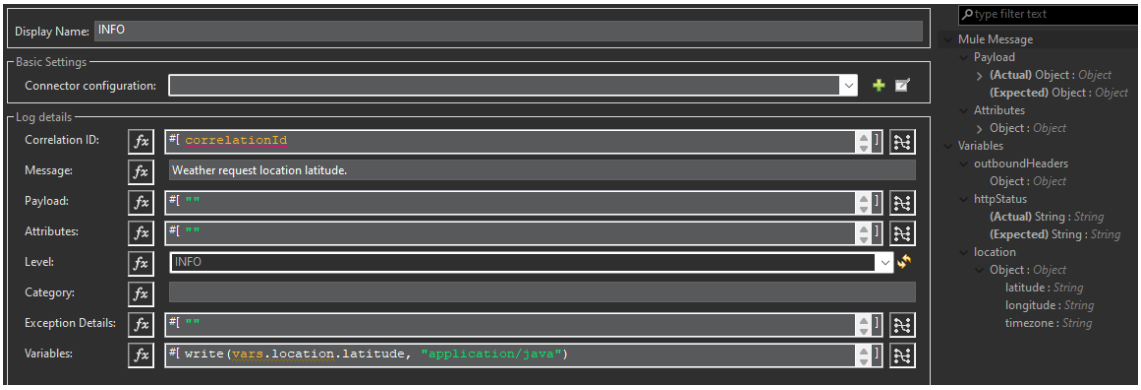

Figura 22 Exemplo 2 de configuração dos parâmetros do logger

Fonte: Elaboração própria

Por último, o exemplo do *logger* colocado no Error Handler. Este é responsável por registar qualquer erro que possa ocorrer durante a execução da aplicação, exibido na Figura 23.

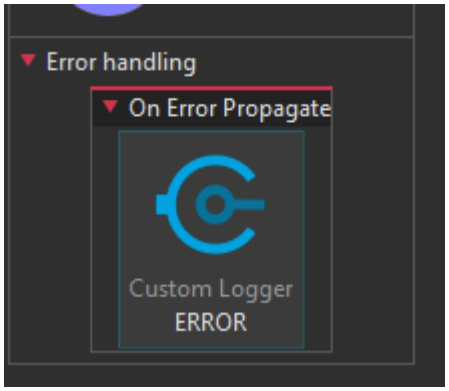

Figura 23 Error handler da aplicação

A configuração mostrada na Figura 24, registar o *id* do evento, o *payload* no momento do erro, os seus atributos, os detalhes do erro e todas as variáveis definidas até ao momento. Neste caso esta informação será registada com o nível *ERROR*. Note-se que o parâmetro *Exception Details* está configurado para caso componente *error* da mensagem seja diferente de nulo, este deve ser registado. O componente error em Mulesoft, agrega todos os detalhes relacionados com o erro ocorrido, tais como localização onde ocorreu, mensagem de erro, o componente que o lançou, entre outros.

Figura 24 Exemplo 3 de configuração dos parâmetros do logger

Fonte: Elaboração própria

7.3 Leitura do Ficheiro no Kibana

O grande objetivo do desenvolvimento deste conector foi possibilitar que os *logs* das aplicações Mulesoft pudessem ser interpretados por *softwares* de visualização e análise de dados, como o Elasticsearch Kibana.

Ao término da instalação e configuração do Elasticsearch, é necessário executar o *bin/elasticsearch*, o *token* de segurança, e a *password* de autenticação serão gerados e exibidos na consola.

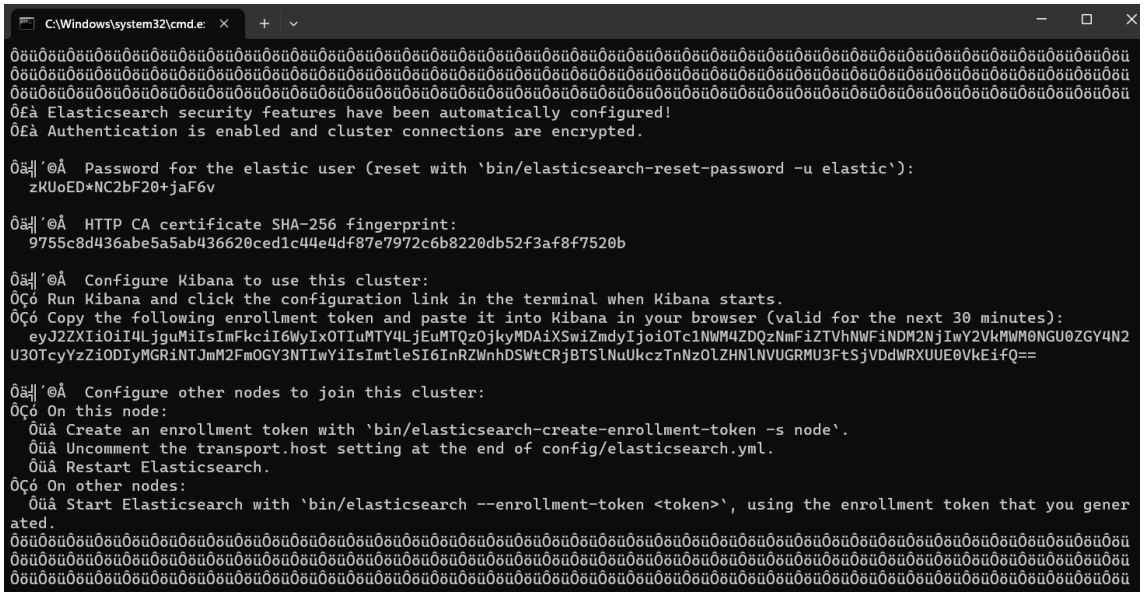

Figura 25 Token Elasticsearch

Fonte: Elaboração própria

Para iniciar o Kibana, apenas é necessário executar o ficheiro *bin\kibana.bat*, presente na pasta do programa, e aceder a *interface* através do browser no endereço *http://localhost:5601*, utilizando o *username* “elastic” e a *password* gerada no passo anterior.

Carregar o ficheiro é o primeiro passo, para isso é necessário ir a *Integrations* e em *Browser integrations* procurar por "*log file*" e selecionar a opção *Upload File*.

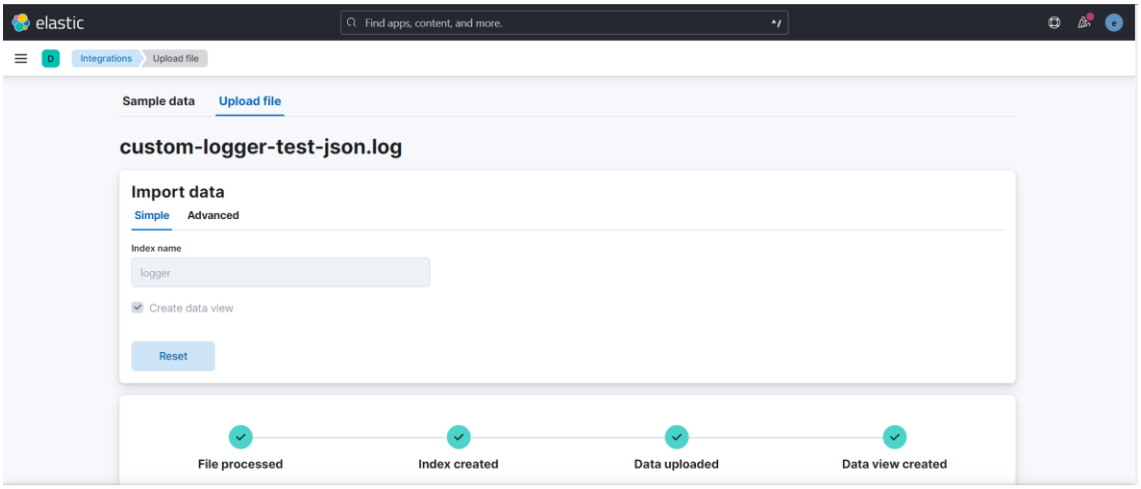

Figura 26 Upload do ficheiro no Kibana

Fonte: Elaboração própria

No final do processo o ficheiro estará carregado, processo mostrado na Figura 26, para permitir a visualização do conteúdo de forma mais fácil, o Kibana conta com a ferramenta Discover, Figura 27. Esta permite explorar e pesquisar sobre dados previamente carregados, na aba do lado esquerdo é possível ver a lista de atributos disponíveis e a direita, a lista de entradas de *log* contidas no documento.

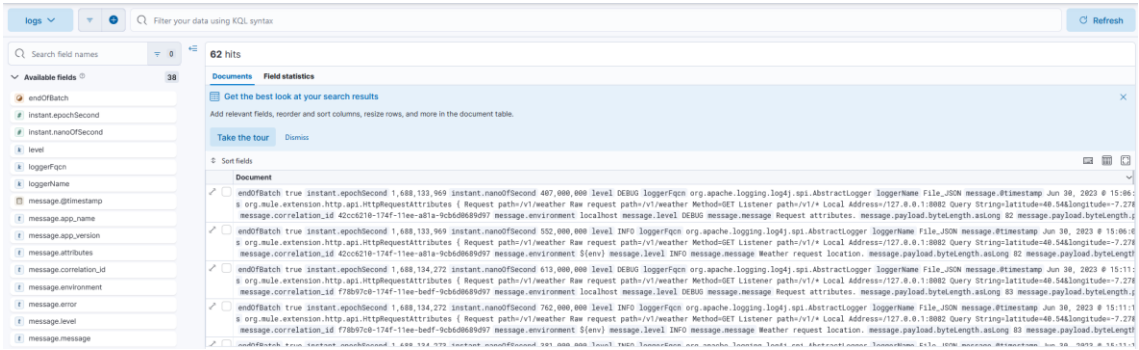

Figura 27 Interface Discover

Fonte: Elaboração própria

Usando o Discover é possível consultar cada parâmetro de cada *log* individualmente, fazer filtros usando KQL, linguagem própria da plataforma, e visualizar valores dos parâmetros em formatos dinâmicos como gráficos e tabelas como pode ser visto no exemplo da Figura 28. Este gráfico foi gerado utilizando os tipos de *log* registados.

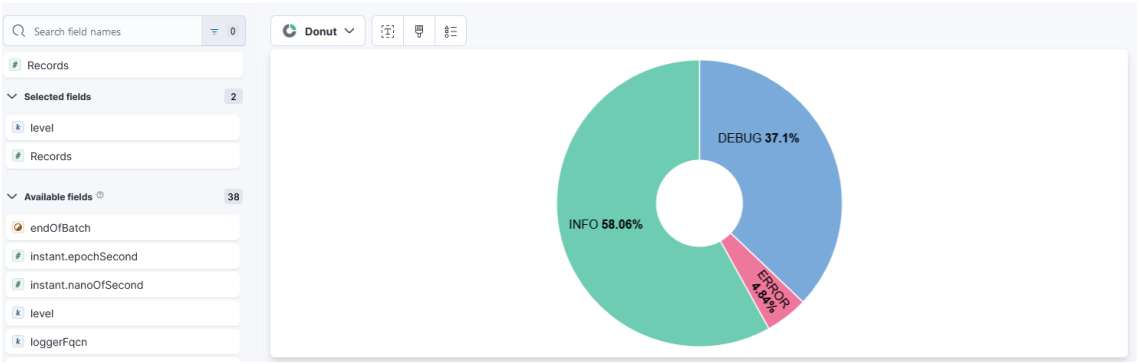

Figura 28 Exemplo de funcionalidade do Discover

Fonte: Elaboração própria

8 Verificação e Validação

O objetivo deste capítulo é validar se o *software* desenvolvido cumpre os requisitos estabelecidos no início do projeto. Para tal, as configurações implementadas anteriormente serão testadas e validadas.

Para fazer pedidos a aplicação, uma coleção de Postman foi criada, de modo a facilitar o processo de testes. Na Figura 29, pode-se observar o pedido feito a aplicação de testes e a resposta obtida. Vários pedidos foram feitos, para fornecer uma quantidade razoável de *logs* para serem visualizados.

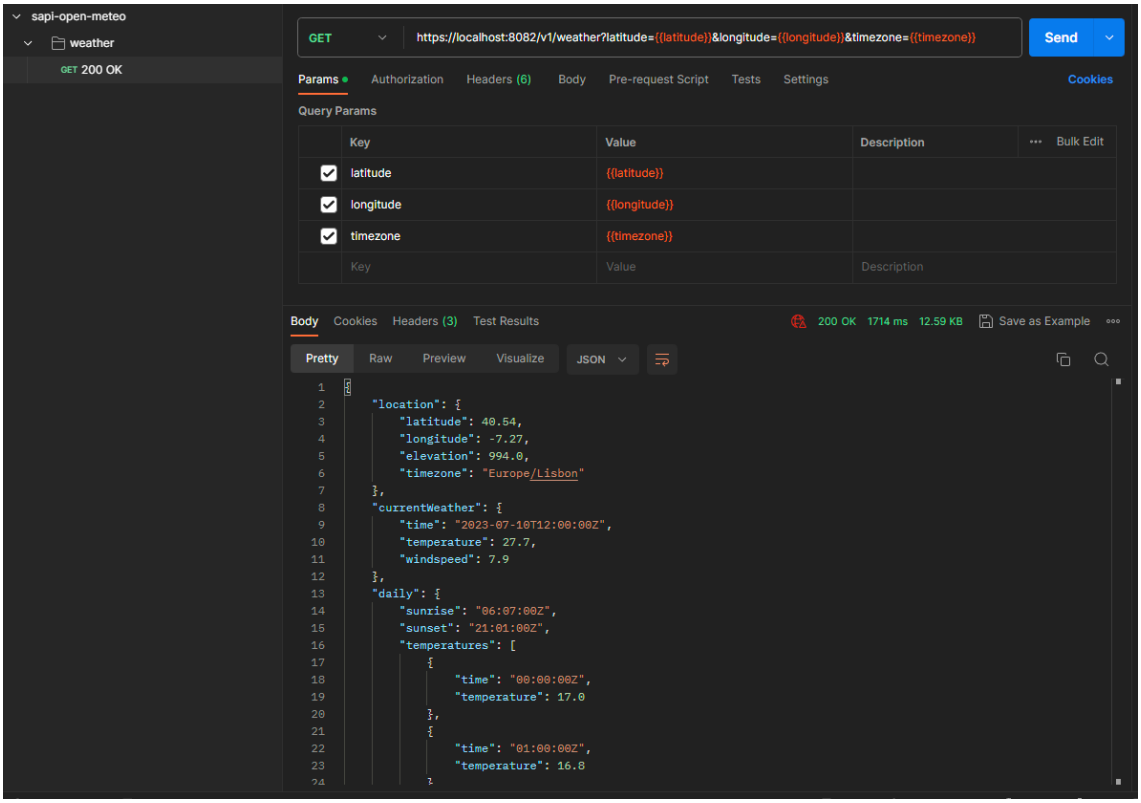

Figura 29 Coleção de testes Postman

Fonte: Elaboração própria

A aplicação Mulesoft, foi executada localmente, de forma a gerar o ficheiro com a informação dos *logs* diretamente no disco. Como mostrado na Figura 30 o ficheiro custom-logger-test-json.log foi gerado e os *logs* dos testes feitos registados pelo conector.

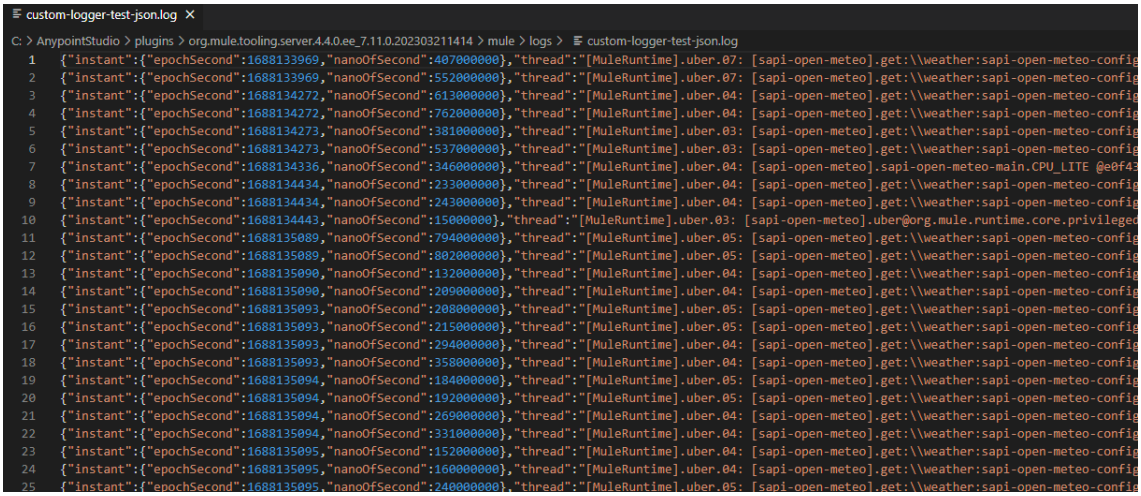

Figura 30 Ficheiro gerado pelo conector

Fonte: Elaboração própria

O ficheiro foi carregado com sucesso pelo Kibana, indicando que o formato dos registos por parte do conector foi o correto. Foi também possível consultar o conteúdo recorrendo à ferramenta Discover, representado na Figura 31.

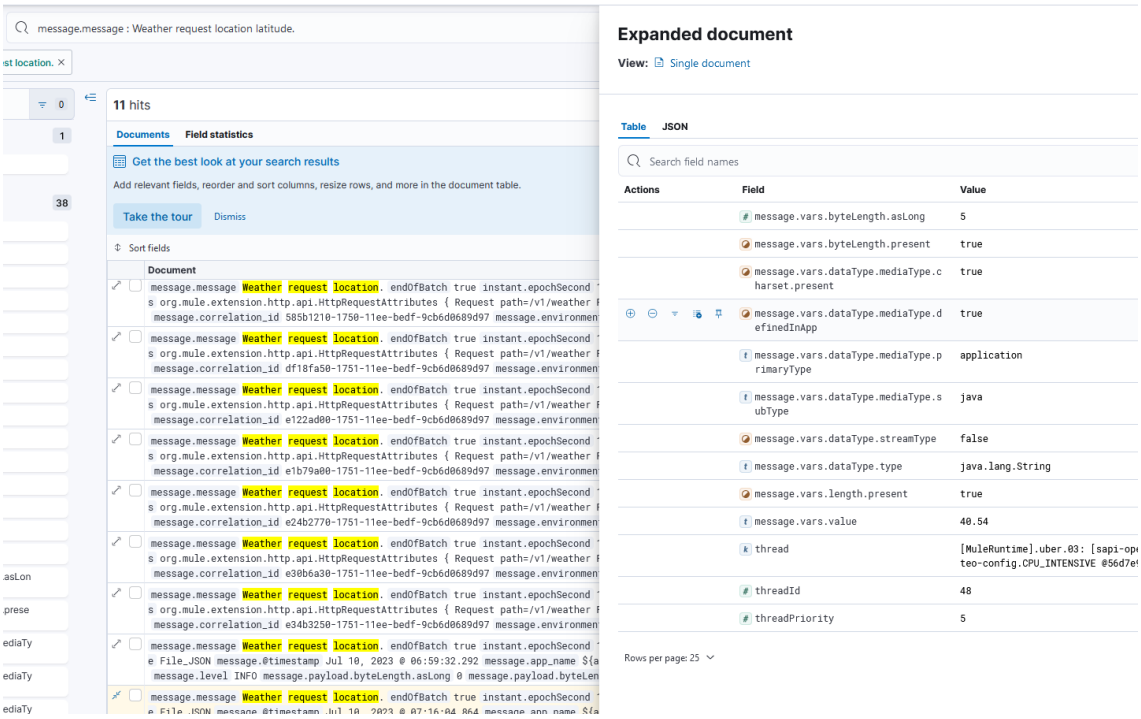

Figura 31 Interface Discover

Fonte: Elaboração própria

Na Figura 31, na aba *Expanded document*, o campo *message.vars.value* indica o valor da latitude, registado durante o teste com a configuração mostrada na Figura 22, indicando ser também possível obter valores de um campo específico de um objeto. Observe-se que um filtro em KQL foi aplicado com o intuído de procurar por um valor específico de um campo.

Em alguns destes testes, foram deliberadamente provocados erros durante execução da aplicação, tais como, *bad requests* não definindo *query parameters* obrigatórios segundo a especificação da API, e a utilização valores de latitude ou longitude não validos.

O conector *logger* inserido no *error handler* da aplicação, configurado com anteriormente representado na Figura 24, pôde registar corretamente o erro mostrado na Figura 32 no campo *message.error*. Neste caso a estrutura de erro completa, sendo que foi constatado uma limitação, não sendo possível registar apenas um parâmetro específico dessa estrutura como, por exemplo, a *description*. Provavelmente devido ao seu formato diferenciado.

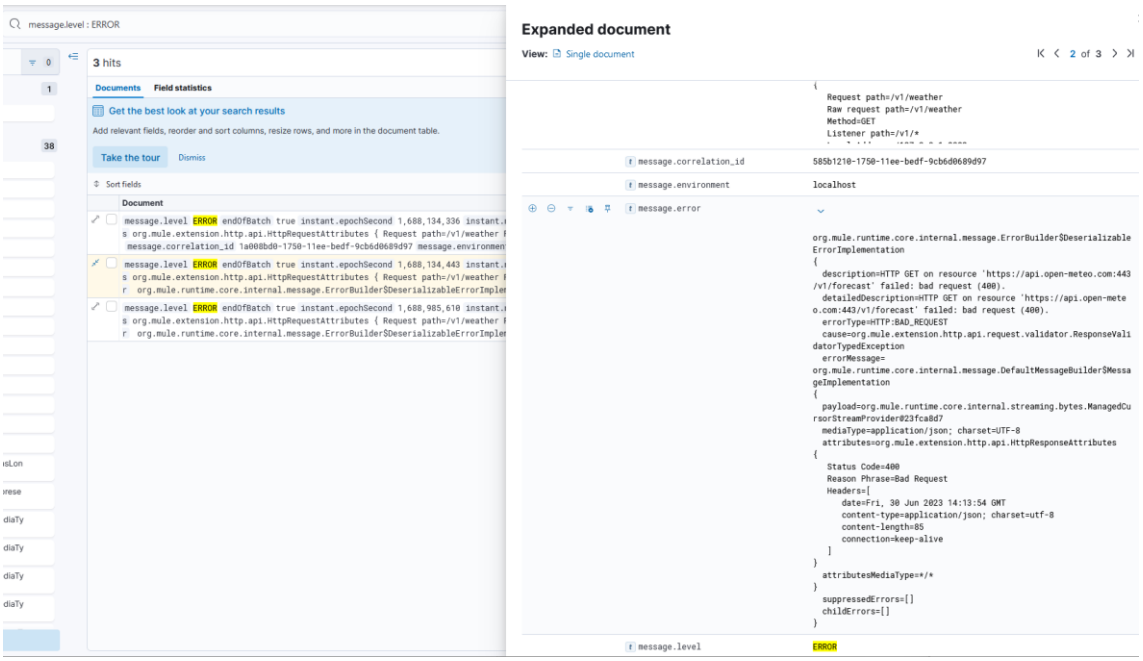

Figura 32 Error message registada pelo logger

Fonte: Elaboração própria

9 Conclusão

O período de estágio foi de extrema importância, não só pelo facto de permitir colocar em prática vários conhecimentos e competências adquiridas no decorrer do percurso académico da Licenciatura de Engenharia Informática, mas também pelo companheirismo e trabalho de equipa vivenciado, que acabou por proporcionar um bom ambiente de trabalho e uma fácil integração. Assim, o estágio curricular foi bastante enriquecedor, como primeiro contacto com o mundo de trabalho, ao permitir que os conhecimentos por vezes pouco sólidos, se consolidassem com a prática.

Desde o início do projeto que o estagiário visou seguir boas práticas como a assiduidade, a pontualidade, a organização, a coordenação do trabalho e a capacidade de adaptação, tentando sempre cumprir e superar os objetivos definidos pela entidade.

A primeira fase do estágio, que consistiu o estudo e autoaprendizagem de novas tecnologias relacionadas com integração de sistemas proporcionou um grande desafio a capacidades de empenho, dedicação e disciplina do estagiário, não só pelo tempo investido na formação, mas também pela complexidade do Mulesoft Fundamentals.

O Mulesoft Develop Fundamentals foi uma formação interessante de se completar, embora se deva admitir que existiram por vezes dificuldades em compreender alguns dos conceitos abordados, não apenas pelo nível de complexidade dos mesmos, mas também pela falta de conhecimentos relacionados com as tecnologias ali exploradas. Embora seja importante referir que a certificação apenas adquirida algum tempo após o término do estágio curricular.

Apesar da formação ter fornecido uma base solida de conhecimentos e funcionamento da plataforma Mulesoft, a preparação e o desenvolvimento do conector personalizado de *logger*, definido como a etapa final do estágio, consistiu em um desafio acima do esperado.

Durante esse processo, o aluno passou por serias dificuldades e desafios no processo de configuração do Maven e adição de dependências e repositórios. Esses desafios acabaram mesmo por estagnar o processo de desenvolvimento por mais de duas semanas, em que o aluno se dedicou a pesquisa de soluções na *internet* para os vários erros que ocorreram durante o processo de *build* e publicação do conector. Chegou mesmo a ser considerada a hipótese de abandonar o projeto, uma vez que nenhum dos membros da equipa tinha conhecimento ou algum tipo de experiência com o desenvolvimento de conectores personalizados. Felizmente, após muita persistência por parte do aluno foi possível encontrar uma forma de resolver os problemas, resultando na solução obtida.

O conector desenvolvido cumpriu com os requisitos propostos, embora tenham sido detetadas algumas limitações de uso. Mesmo os parâmetros do conector podendo ser definidos com recurso ao Dataweave, em algumas situações não foi possível resgatar apenas um ou mais campos específicos do parâmetro a ser configurado. Sendo assim, uma situação que teria de ser considerada em futuros melhoramentos.

Na opinião do aluno, este foi um estágio que agregou não só uma grande variedade de conhecimentos em várias tecnologias, mas também uma formação que decerto será muito útil para o futuro profissional. Assim como, a lição de que por mais dificuldades que a apareçam é muito importante manter o empenho e dedicação, dessa forma todos os objetivos poderão ser alcançados.

Referências

Apache Software Foundation. 2004. WHAT IS CAMEL? *Camel Apache.* [Online] 2004. https://camel.apache.org/manual/faq/what-is-camel.html.

—. What is Maven? *Maven Apache.* [Online] https://maven.apache.org/what-is-maven.html.

Bafna, Jitendra. 2021. What is MuleSoft and Anypoint Platform Capabilities and Strengths. *DZone.* [Online] 15 de Agosto de 2021. https://dzone.com/articles/what-is-mulesoft-and-anypoint-platform-capabilitie.

Consulting, Smart. 2022. Smart Consulting. *Metodologia Ágil × Tradicional: Afinal, qual é a diferença?* [Online] 2022. https://blog.smartconsulting.com.br/metodologia-agil-x-tradicional/.

Elasticsearch B.V. What is Elasticsearch? *Elastic.* [Online] https://www.elastic.co/what-is/elasticsearch.

Erney, Joshua. 2022. What is DataWeave? Part 1: The Basics. *Developer Mulesoft.* [Online] Dezembro de 2022. https://developer.mulesoft.com/tutorials-and-howtos/dataweave/what-is-dataweave-getting-started-tutorial/.

Espiñeira, J. D. 2021. Metodologías Ágiles: Qué es Scrum? *Proactive Office.* [Online] 2021.

IBM. 2015. IBM Integration Bus. *IBM Documentation Help.* [Online] 2015. https://www.ibm.com/docs/en/integration-bus/10.0?topic=overview-integration-bus-introduction.

Machado, F. N. R. 2018. *Análise e Gestão de Requisitos e Software Onde nascem ossistemas. 3.ª ed.* São Paulo : Saraiva Educação, 2018.

Madaan, Jeetan. 2020. Introduction to Mule SDK: Part 1. *APISero.* [Online] 7 de Julho de 2020. https://apisero.com/introduction-to-mule-sdk-part-1/.

Mulesoft. Anypoint Studio. *Mulesoft.* [Online] https://www.mulesoft.com/pt/platform/studio.

Mulesoft Docs. Introduction to Mule 4: Mule Message. *Mulesoft Docs.* [Online] https://docs.mulesoft.com/mule-runtime/4.4/intro-mule-message.

—. Mule SDK: Learning the Module Model. *Mulesoft Docs.* [Online] https://docs.mulesoft.com/mule-sdk/1.1/module-structure.

Mulesoft from Salesforce. Anypoint Platform Development: Fundamentals. *Training Mulesoft.* [Online] https://training.mulesoft.com/course/development-fundamentals-mule4.

—. Introduction to Anypoint Connectors. *Mulesoft Docs.* [Online] https://docs.mulesoft.com/connectors/introduction/introduction-to-anypoint-connectors.

Mulesoft. Logger Component. *Docs Mulesoft.* [Online] https://docs.mulesoft.com/mule-runtime/4.4/logger-component-reference.

—. Publish assets Using Maven. *Docs Mulesoft.* [Online]

—. Tutorial: Build an API from Start to Finish. *Docs Mulesoft.* [Online] https://docs.mulesoft.com/general/api-led-overview.

Munday, Jess. 2022. What Is MuleSoft? *Salesforce Blog.* [Online] Salesforce Inc, 15 de Setembro de 2022. https://www.salesforce.com/blog/what-is-mulesoft/.

O'Neill, Luke. 2021. Microsoft Teams. *TechTarget.* [Online] Março de 2021. https://www.techtarget.com/searchunifiedcommunications/definition/Microsoft-Teams.

Open-Meteo.com. Free Weather API. *Open Meteo.* [Online]

Oracle. 2023. Integration Services. *Oracle Integration.* [Online] 2023. https://www.oracle.com/in/integration/.

—. What is Java? *Java.* [Online] https://www.java.com/en/download/help/whatis_java.html.

Paula, G. B. 2016. Tudo sobre Metodologia Scrum: o que é e como essa ferramenta. *Treasy.* [Online] 2016. https://www.treasy.com.br/blog/scrum/.

Pedamkar, Priya. 2023. What is Visual Studio Code? *EDUCBA.* [Online] 13 de Março de 2023. https://www.educba.com/what-is-visual-studio-code/.

Prashant Choudhary. 2023. What Is API-led Connectivity? Unlock Business Agility. *The 360 Blog.* [Online] 20 de Abril de 2023.

Racios. 2022. Merkletech, Unipessoal Lda. *Racios.* [Online] 2022. https://www.racius.com/merkletech-unipessoal-lda/.

Salesforce Trailhead. Trailhead. *Salesforce Trailhead.* [Online]

Sritter. 2021. What is OpenJDK? *Azul.* [Online] 15 de Setembro de 2021. https://www.azul.com/blog/what-is-openjdk/.

Anexos

Anexo 1 – Classes do Connector

CustomLoggerConfiguration

```
/**
 * This class represents an extension configuration, values set in
this class are commonly used across multiple
 * operations since they represent something core from the extension.
 */
@Operations(CustomLoggerOperations.class)
@ConnectionProviders(CustomLoggerConnectionProvider.class)
public class CustomLoggerConfiguration {

    @Parameter
    @DisplayName("App Name")
    @Summary("The name of the application")
    @Optional(defaultValue = "${app.name}")
    private String appName;

    @Parameter
    @DisplayName("App Version")
    @Summary("The version of the application")
    @Optional(defaultValue = "${app.version}")
    private String appVersion;

    @Parameter
    @DisplayName("Environment")
    @Summary("App environment")
    @Optional(defaultValue = "${env}")
    private String environment;

    public String getAppName() {
        return appName;
    }

    public String getAppVersion() {
        return appVersion;
    }
```

```
    public String getEnvironment() {
      return environment;
    }
}
```

CustomLoggerConnection

```
/**
 * This class represents an extension connection just as example
(there is no real connection with anything here c:).
 */
public final class CustomLoggerConnection {

  private final Logger LOGGER = (Logger)
LogManager.getLogger("File_JSON");

  public Logger getLOGGER(){
    return LOGGER;
  }

  public CustomLoggerConnection() {

  }

  public void invalidate() {
    // do something to invalidate this connection!
  }
}
```

CustomLoggerConnectionProvider

```
public class CustomLoggerConnectionProvider implements
PoolingConnectionProvider<CustomLoggerConnection> {

  @Override
  public CustomLoggerConnection connect() throws ConnectionException {
    return new CustomLoggerConnection();
  }

  @Override
  public void disconnect(CustomLoggerConnection connection) {
    try {
      connection.invalidate();
    } catch (Exception e) {
       e.getMessage();
    }
  }

  @Override
  public ConnectionValidationResult validate(CustomLoggerConnection
connection) {
    return ConnectionValidationResult.success();
  }
}
```

CustomLoggerExtension

```
/**
 * This is the main class of an extension, is the entry point from
which configurations, connection providers, operations
 * and sources are going to be declared.
 */
@Xml(prefix = "custom-logger")
@Extension(name = "Custom Logger")
@Configurations(CustomLoggerConfiguration.class)
public class CustomLoggerExtension {


}
```

CustomLoggerOperations

```
public class CustomLoggerOperations {

  @MediaType(value = ANY, strict = false)
  @DisplayName("Custom Logger File")

  public void createLoggerFile(@Config CustomLoggerConfiguration
configuration, @Connection CustomLoggerConnection connection,
@ParameterGroup (name = "Log details") CustomLoggerParameters
logParams, CompletionCallback <Void, Void> callback) {
    try {

      Logger logger = connection.getLOGGER();

      System.out.println("Starting Log: " + logger);

      Map <String, Object> logMsg = new HashMap<String, Object>();

      //time
      logMsg.put("@timestamp", Instant.now().toString());

      //App Context
      logMsg.put("app_name", configuration.getAppName());
      logMsg.put("app_version", configuration.getAppVersion());
      logMsg.put("environment", configuration.getEnvironment());

      //Message context
      logMsg.put("level", logParams.getLevel());
      logMsg.put("correlation_id", logParams.getCorrelationId());
      logMsg.put("category", logParams.getCategory());
      logMsg.put("message", logParams.getMessage());
      logMsg.put("payload", logParams.getPayload());
      logMsg.put("attributes", logParams.getAttributes());
      logMsg.put("exception", logParams.getError());
      logMsg.put("vars", logParams.getVars());

      ObjectMessage obj = new ObjectMessage(logMsg);
      logger.log(getLogLevel(logParams.getLevel()), obj);
```

```
      System.out.println("Completed Log: " + logger);
      System.out.println("Log path:
C:\\AnypointStudio\\plugins\\org.mule.tooling.server.4.3.0.ee_7.3.5.20
2108031239\\mule\\logs");

      callback.success(Result.<Void, Void>builder().build());

    } catch (Exception e) {
      callback.error(e);
    }
  }

  private static Level getLogLevel(String level) {
    Level levelValue;

    switch (level) {
      case "TRACE":
        levelValue = Level.TRACE;
        break;
      case "DEBUG":
        levelValue = Level.DEBUG;
        break;
      case "INFO":
        levelValue = Level.INFO;
        break;
      case "WARN":
        levelValue = Level.WARN;
        break;
      case "ERROR":
        levelValue = Level.ERROR;
        break;
      case "FATAL":
        levelValue = Level.FATAL;
        break;
      default:
        levelValue = Level.INFO;
        break;
    }
    return levelValue;
  }
```

```
}
```

LogLevel

```
public class LogLevel implements ValueProvider{
    @Override
    public Set<Value> resolve() {
        return
ValueBuilder.getValuesFor("DEBUG","ERROR","INFO","TRACE","FATAL","WARN
");
    }
}
```

CustomLoggerParameters

```
public class CustomLoggerParameters {

    @Parameter
    @Optional(defaultValue = "#[correlationId]")
    @DisplayName("Correlation ID")
    @Summary("Correlation UUID")
    @Example("#[correlationId]")
    private String correlationId;

    @Parameter
    @Optional
    @DisplayName("Message")
    @Summary("Message to be logged")
    private String message;

    @Parameter
    @Optional(defaultValue = "#[write(payload,
\"application/java\")]")
    @DisplayName("Payload")
    @Summary("Payload to be logged")
    @Example("#[write(payload, \"application/java\")]")
    private TypedValue<Object> payload;

    @Parameter
        @Optional(defaultValue = "#[write(attributes,
\"application/java\")]")
        @DisplayName("Attributes")
        @Summary("Attributes to be logged")
        @Example("#[write(attributes, \"application/java\")]")
        private String attributes;

    @Parameter
    @DisplayName("Level")
    @OfValues(LogLevel.class)
    @Optional(defaultValue = "INFO")
    private String level;

    @Parameter
```

```
    @Optional
    @DisplayName("Category")
    private String category;

    @Parameter
    @Optional(defaultValue = "#[if (error != null) write(error,
\"application/java\") else \"\"]")
    @DisplayName("Exception Details")
    @Summary("Exception to be logged")
    @Example("#[if (error!= null) write(error, \"application/java\")
else \"\"]")
    private String error;

    @Parameter
    @Optional(defaultValue = "#[vars]")
    @DisplayName("Variables")
    @Example("#[vars]")
    private Map<String, TypedValue<Object>> vars;

    public String getCorrelationId() {
            return correlationId;
      }

      public String getMessage() {
            return message;
      }

      public TypedValue<Object> getPayload() {
            return payload;
      }

      public String getAttributes() {
            return attributes;
      }

      public String getLevel() {
            return level.toString();
      }

      public String getCategory() {
            return category;
```

```
	}

	public String getError() {
		return error;
	}

	public Map<String, TypedValue<Object>> getVars() {
		return vars;
	}
}
```

Anexo 2 - Configurações do ficheiro POM

Configuração *Plugin* Maven

```
<build>
            <pluginManagement>
                  <plugins>
                        <plugin>
            <groupId>org.mule.tools.maven</groupId>
            <artifactId>exchange-mule-maven-plugin</artifactId>
            <version>0.0.14</version>
            <executions>
                <execution>
                    <id>validate</id>
                    <phase>validate</phase>
                    <goals>
                        <goal>exchange-pre-deploy</goal>
                    </goals>
                </execution>
                <execution>
                    <id>deploy</id>
                    <phase>deploy</phase>
                    <goals>
                        <goal>exchange-deploy</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
                  </plugins>
            </pluginManagement>
      </build>
```

Perfil repositório Exchange

```
<profiles>
   <profile>
       <id>exchange</id>
       <distributionManagement>
           <repository>
               <id>ExchangeRepository</id>
               <name>Corporate Repository</name>
<url>https://maven.anypoint.mulesoft.com/api/v1/organizations/${projec
t.groupId}/maven</url>
               <layout>default</layout>
           </repository>
       </distributionManagement>
   </profile>
</profiles>
```

Repositório Maven

```
<pluginRepositories>
        <pluginRepository>
            <id>mulesoft-releases</id>
            <name>mulesoft release repository</name>
            <layout>default</layout>
            <url>https://repository.mulesoft.org/releases/</url>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
        </pluginRepository>
 </pluginRepositories>
```

Anexo 3 – get-weather-response.raml

```
#%RAML 1.0 DataType

properties:
  location:
    properties:
      latitude:
        type: number
        example: 52.52
      longitude:
        type: number
        example: 13.419
      elevation:
        type: number
        example: 44.812
      timezone:
        example: "Europe/Berlin"
  currentWeather:
    properties:
      time:
        type: datetime
      temperature:
        type: number
        example: 13.3
      windspeed:
        type: number
        example: 10.3
  daily?:
    properties:
      sunrise?:
        type: time-only
      sunset?:
        type: time-only
      temperatures?:
        type: array
        items:
          properties:
            time:
              type: time-only
            temperature:
              type: number
              example: 23.2
```

Anexo 4 – localhost.yaml

```yaml
#keystore
keystore:
  path: "keystore.jks"

#app
app:
  name: "sapi-open-meteo"
  version: "1.0.0"

#open-meteo
open-meteo:
  host: "api.open-meteo.com"
  path: "/v1"
```

Anexo 5 - Postman collection


```
{
	"info": {
		"_postman_id": "15bd3936-f2d9-4e7b-bccb-246292ebd392",
		"name": "sapi-open-meteo",
		"schema":
"https://schema.getpostman.com/json/collection/v2.1.0/collection.json"
,
		"_exporter_id": "28271241"
	},
	"item": [
		{
			"name": "weather",
			"item": [
				{
					"name": "200 OK",
					"request": {
						"method": "GET",
						"header": [],
						"url": {
							"raw":
"https://localhost:8082/v1/weather?latitude={{latitude}}&longitude={{l
ongitude}}&timezone={{timezone}}",
							"protocol": "https",
							"host": [
								"localhost"
							],
							"port": "8082",
							"path": [
								"v1",
								"weather"
							],
							"query": [
								{
									"key":
"latitude",
									"value":
"{{latitude}}"
								},
								{
```

```
                                                    "key":
"longitude",
                                                    "value":
"{{longitude}}"
                                                },
                                                {
                                                    "key":
"timezone",
                                                    "value":
"{{timezone}}"
                                                }
                                            ]
                                        }
                                    },
                                    "response": []
                                }
                            ]
                        }
                ]
}
```