UNIVERSIDADE FEDERAL DO ABC

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ESTÁGIO SUPERVISIONADO III

LUCAS ZANFERRARI CARAÇA

Orientador: Prof. Dr. Thiago Ferreira Covões

Santo André – SP

2019

LUCAS ZANFERRARI CARAÇA

ESTÁGIO I: DATA INTEGRATOR NA KEYRUS BRASIL
ESTÁGIOS II E III: DESENVOLVEDOR DE SOFTWARE SENIOR NA AVENUE CODE

Relatório de estágio apresentado ao Centro de Matemática, Computação e Cognição e à coordenação do Bacharelado em Ciência da Computação da Universidade Federal do ABC como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Thiago Ferreira Covões

Lucas Zanferrari Caraça	Thiago Ferreira Covões
Discente	Professor Orientador

09 de Maio de 2019, Santo André – SP

DEDICATÓRIA

Aos professores e mentores que guiaram minha trajetória, e aos colegas profissionais que tive o prazer de colaborar durante esta árdua e edificante jornada.

AGRADECIMENTOS

Ofereço meus sinceros agradecimentos:

À minha família e amigos(as), pelo apoio incondicional a mim dedicado.

Ao professor orientador Thiago Ferreira Covões, pelos conselhos acadêmicos e por guiar a escrita deste trabalho.

À todos da Keyrus Brasil, pela oportunidade e crença em meu potencial.

À todos da Avenue Code, também pela oportunidade, e pelo crescimento profissional a mim proporcionado.

À todos da Anheuser-Busch InBev, pelos valores ensinados e pela excelência profissional.

À todos da MuleSoft, por me receberem de braços abertos desde o primeiro dia e contribuírem significativamente para meu amadurecimento técnico e pessoal.

"In order to change an existing imagined order, we must first believe in an alternative imagined order."

Harari [14]

Resumo

Este texto tem como objetivo principal descrever as atividades realizadas como *Data Integrator* na Keyrus Brasil, uma consultoria de Tecnologia da Informação centrada majoritariamente na área de *Business Intelligence*, bem como as exercidas como Desenvolvedor de Software Senior na Avenue Code, também uma provedora de serviços de consultoria de Tecnologia da Informação, mas com foco em construção e manutenção de plataformas de *software* sob encomenda. Com esta finalidade, as empresas serão apresentadas. A fundamentação de conhecimento necessária para acompanhá-lo também será abordada, assim como a motivação por trás das tarefas executadas no exercício dos cargos, e os escopos das atribuições dos cargos em si.

As atividades descritas neste relatório tiveram ocorrência durante o Bacharelado em Ciências da Computação na Universidade Federal do ABC. As relativas ao Estágio Supervisionado I ocorreram de junho de 2017 até novembro de 2018, enquanto as relativas ao Estágio Supervisionado II ocorreram entre fevereiro de 2019 e março de 2019 e as relacionadas ao Estágio Supervisionado III, entre abril de 2019 e maio de 2019.

O relatório detalhará a intersecção entre o conteúdo abordado no curso e a demanda do trabalho, mencionando as disciplinas que tiveram maior relevância na composição desta intersecção, e evidenciará o conteúdo extra que foi requerido para que a execução das atividades em ambos os cargos se tornasse viável.

Palavras-Chave: Business Intelligence, SCRUM, Engenharia de Software, REST, Big Data, ETL, Python, Integração de Sistemas, API-Led Connectivity, Mule ESB, Azure, CloudHub.

Abstract

This text has as its main goal to describe the activities carried out as Data Integrator at Keyrus Brasil, an Information Technology consultancy majorly centered in the field of Business Intelligence, as well as the ones exerted as Senior Software Developer at Avenue Code, also an Information Technology consulting services provider. but focusing on building and maintaining custom software platforms. For this purpose, the companies will be presented. The knowledge base that is required to follow it will be addressed, as well as the motivation behind the tasks executed during both positions' exercise, and the scopes of the duties of the positions themselves. The activities hereby described occured while enrolled in the Federal University of ABC's Bacharelor in Computer Sciences program. The ones related to the Supervised Internship I occurred from June of 2017 until November of 2018, while the ones related to the Supervised Internship II occurred between February of 2019 and March of 2019 and the ones relative to the Supervised Intership III, between April of 2019 and May of 2019.

The report will detail the intersection between the course content and the actual work demand, mentioning the subjects that had the most relevance in the intersection's composition, and will detail the extra content that was required in order to make the tasks' execution in both positions viable.

Keywords: Business Intelligence, SCRUM, Software Engineering, REST, Big Data, ETL, Python, Systems Integration, API-Led Connectivity, Mule ESB, Azure, CloudHub.

Sumário

Lista de Figuras

1 Introdução (Keyrus Brasil)

Nesta seção, serão apresentados os aspectos organizacionais da empresa Keyrus, além de uma visão funcional do trabalho executado. O vínculo empregatício ocorreu desde junho de 2017 até a fevereiro de 2019. O principal objetivo deste consistiu no *design* da arquitetura de dados e construção dos procedimentos de extração, tratamento e ingestão automática de dados, caracterizados como aplicações de ETL (*Extract, Transform and Load*) na terminologia técnica, de uma plataforma de monitoramento de KPIs (*Key Performance Indicators*) - denominada *Commercial Insights Platform*, ou CIP - relacionados ao desempenho das campanhas de *marketing* da empresa cliente, a Anheuser-Busch InBev, cervejaria proprietária do maior número de marcas de cervejas, entre outros produtos, no mundo.

Além disso, manutenções ocasionais em aplicações de ETL de outros sistemas também ocorreram e fizeram parte da função.

1.1 Instituição concedente: Keyrus Brasil

Endereço: Av. Paulista, 1374, 6° andar - Bela Vista.

CEP: 01310-100

Cidade: São Paulo

Estado: SP

CNPJ: 05.341.639/0001-54

Representado por: Anselmo Ressel **Cargo:** Diretor Executivo de *Delivery*

Supervisor de Estágio: Anselmo Ressel

Horário de trabalho: 09h às 18h, com 1h de almoço

Total semanal: 40 horas.

Atividades:

- Definição e implementação da arquitetura de dados que será consumida pelas funcionalidades do sistema CIP;
- Definição e implementação das aplicações de ETL que fazem extração, validação, transformação e carga dos dados automática no *Data Warehouse* do CIP, por vezes utilizando os recursos do *Data Lake* (um *cluster* Hadoop [20]) contido na infraestrutura do sistema;
- Contribuição para com as cerimônias previstas pela metodologia ágil SCRUM, a qual é utilizada para gerenciar a execução do projeto, tais como reuniões diárias, revisão da *sprint*, planejamento

da sprint e *planning poker*;

- Participação no treinamento de metodologia ágil com o professor José Finocchio Junior, criador da metologia Canvas.

1.1.1 Objetivos a serem alcançados

Durante o período de atuação, o esperado é que o colaborador planeje, implemente, documente e comunique as funcionalidades tangentes a infraestrutura de dados do sistema.

Além disso, espera-se que ele interaja com os analistas de negócios, testadores, gerentes de *marketing*, *SCRUM master*, *Product Owner* e desenvolvedores das visualizações de dados para entender as formas de cálculo de cada KPI, o que o cliente espera da funcionalidade sendo desenvolvida do ponto de vista funcional, facilitar o consumo dos dados pelos desenvolvedores das visualização e, ao final, entender e corrigir inconsistências entre os valores esperados dos KPIs e os valores exibidos pelo sistema.

1.1.2 Caracterização e análise da Keyrus

A Keyrus é uma consultoria de tecnologia da informação fundada em 1996. Está presente em 17 países, 4 continentes, e possui como principais serviços o desenvolvimento de plataformas digitais, sistemas de *Business Intelligence* (ou BI), aplicações de *Big Data*, *Data Science* e/ou *Internet of Things*.

Sua sede se encontra na França, mas o segundo país mais relevante, devido ao tamanho da fatia de lucro líquido aqui gerada, é o Brasil. A matriz brasileira encontra-se em São Paulo, no sexto andar do WeWork Paulista - um espaço de *coworking* cuja marca é referência mundial na área - mas a empresa também possui escritórios em Campinas, Florianópolis e Belo Horizonte.

Até dezembro de 2018, a empresa contava com uma equipe multidisciplinar de aproximadamente 700 funcionários, dos quais pouco mais de 200 fazem parte da *Business Unit* de *Big Data*, a qual fornece o cargo de *Data Integrator*. Na Figura 1, apresenta-se o logotipo da Keyrus.

Figura 1: Logotipo da Keyrus.

As principais áreas de atuação são:

- *Business Intelligence*
- *Digital Transformation*
- *Big Data*
- *Data Science*
- *Internet of Things*
- Metologia ágil aplicada a gerência de projetos

A Keyrus Brasil atende diversos clientes notórios no mercado brasileiro. Alguns exemplos são:

- Porto Seguro
- Anheuser-Busch InBev
- Claro
- NET
- Vivo
- Grupo Pão de Açúcar
- Telefônica

1.1.3 Características da área onde o trabalho foi realizado

Os trabalhos foram realizados, em sua maioria, remotamente, o que compreende cafés, restaurantes e espaços de convivência, mas na maior parte dos dias, o escritório domiciliar do colaborador. Para o caso específico das atividades desempenhadas por este colaborador, o escritório na Avenida Paulista foi utilizado apenas para ocasionais reuniões de projetos e negócios. Isto também ocorreu, em determinadas situações, no escritório do cliente que o colaborador prestou serviço no decorrer do período das atividades (AB InBev), localizado em Jaguariúna.

A Keyrus Brasil aluga um andar inteiro dentro do WeWork Paulista para seus colaboradores utilizarem. Este andar conta com copa própria, mais de uma dezena de salas de reuniões, mesas (ao invés de baias, para estimular a colaboração) com computadores utilizados pelos funcionários, além de uma sala de descompressão (com jogos de mesa como bilhar e pebolim).

O escritório da AB InBev encontra-se em Jaguariúna, no endereço Av. Antártica, 1353, e também possui as mesmas facilidades. Toda vez que o deslocamento até o escritório do cliente se fez necessário, a Keyrus Brasil assumiu a responsabilidade dos custos envolvidos por meio do reembolso ao colaborador.

2 Introdução (Avenue Code)

A seguir a empresa Avenue Code será discutida. Sua organização será apresentada e as atividades que compõe o escopo do cargo serão listadas e descritas. O vínculo empregatício com a empresa ocorre desde fevereiro de 2019 até o presente momento (maio de 2019). O seu objetivo consiste no *design* da arquitetura e implementação de integrações de sistemas por meio da criação de RESTful APIs (*Application Programming Interface* e RESTful, a sua característica de se respeitar um padrão de *design* de aplicações chamado *Representational State Transfer*), *event-driven APIs* (APIs baseadas em eventos, usualmente processados por *Message Brokers* e frequentemente utilizadas em aplicações de tempo real) e integrações de processamento de dados em *batches* utilizando a ferramenta Mule ESB. Esta ferramenta foi desenvolvida pela MuleSoft, uma empresa pertencente a SalesForce cujo principal objetivo é criar ferramentas que agilizem a transformação digital de seus clientes por meio da facilitação da integração entre seus sistemas e/ou sistemas de terceiros. As ferramentas da MuleSoft são comercializadas com seus clientes por meio de um serviço de assinatura, e a Avenue Code é uma parceira oficial desta, de forma que é certificada para oferecer mão-de-obra qualificada para a execução de projetos de integração de *software* utilizando suas tecnologias.

2.1 Instituição concedente: Avenue Code

Endereço: Rua Luís Coelho, 223, 3° e 8° andares - Consolação.

CEP: 01309-001

Cidade: São Paulo

Estado: SP

CNPJ: 12.131.444/0002-26

Representado por: Ricardo Ribeiro **Cargo:** *Technical Manager*

Supervisor de Estágio: Ricardo Ribeiro

Horário de trabalho: 10h às 19h, com 1h de almoço

Total semanal: 40 horas.

Atividades:

- Coleta de requisitos com o cliente através da execução de repetidas reuniões de planejamento;
- Definição da arquitetura dos sistemas que serão responsáveis por integrar diversos outros sistemas;
- Definição do contrato de cada rota (ou *endpoint*) das APIs a serem desenvolvidas utilizando o a linguagem de especificação RAML [6];

- Desenvolvimento da arquitetura proposta utilizando Mule ESB e suas tecnologias (linguagem DataWeave, seu editor visual denominado Anypoint Studio, seu *message broker* de nome AnypointMQ e, por vezes, a linguagem Java para transformações de dados não suportadas pelas alternativas previamente citadas);
- *Deployment*, ou implantação, das APIs ou aplicações de integração desenvolvidas no serviço de computação em nuvem da MuleSoft, chamado CloudHub;
- Participação em palestras de capacitação profissional quinzenais;
- Conclusão de um curso online de Mule 4, a mais nova (e repleta de funcionalidades diferentes) versão do *suite* de produtos da MuleSoft.

2.1.1 Objetivos a serem alcançados

O cargo ocupado pelo colaborador possui como atribuições gerais as seguintes: planejamento, implementação, documentação e comunicação das APIs que compõe a rede de aplicações responsável por integrar os sistemas corporativos do cliente e expô-los por meio de interfaces de mesmo formato, como ditam as regras que definem um *Enterprise Service Bus* (ESB) [12].

Espera-se que a rede de aplicações siga o padrão de projetos evangelizado pela MuleSoft, que por sua vez é denominado *API-Led Connectivity*. Este padrão de projetos será discutido em maior nível de detalhamento mais adiante no texto. Também é preciso que os trechos de software desenvolvidos sejam pautados em boas práticas gerais da indústria, como modularização de funcionalidades, desacoplamento de escopo sempre que possível, criação da cobertura de testes unitários onde aplicável e assim por diante, do contrário, o que foi construído pode não ser incorporado de fato a rede de aplicações após passar um processo de revisão de código.

Além disso, espera-se que o ocupante do cargo interaja com *Product Owners*, *SCRUM Masters*, usuários, gerentes de projetos e outros desenvolvedores das equipes antes de efetuar as entregas. O intuito desta prática é garantir que o trabalho em desenvolvimento atenderá as expectativas funcionais dos *Stakeholders*, que frequentemente são os clientes da consultoria.

2.1.2 Caracterização e análise da Avenue Code

A Avenue Code é uma consultoria de especializada na construção de produtos de software sob demanda. Ela foi fundada em 2008 e possui atuação em 3 países: Canadá, Estados Unidos e Brasil. Possui escritórios estabelecido em em Montreal, São Francisco, Nova Iorque, São Paulo e Belo Horizonte. Sua sede fica na última, porém a maior parte de sua carteira de clientes é estadunidense.

Toda plataforma entregue pela Avenue Code é guiada pelas melhores práticas da indústria e a empresa têm sido reconhecida por isso no decorrer dos anos. Para que isso seja possível, funcionários são encorajados a se capacitarem, além de compartilharem o conhecimento por meio de treinamentos internos. Estes treinamentos internos ocorrem em horário de trabalho e são frequentes.

A empresa também se posiciona como contribuinte em ações sociais por meio de um programa interno chamado AC Social. Este programa visa capacitar estudantes economicamente carentes por meio do fornecimento de cursos elaborados e ministrados por seus colaboradores, assim como a tradução do conteúdo da plataforma Code.org para o português brasileiro.

Outro diferencial relevante da empresa é a política interna *Zero Means Zero* (ZMZ) O seu intuito é reforçar que nenhuma ação de assédio sexual ou moral, sexismo, racismo, homofobia, preconceito social, político ou econômico ou recriminação por incapacidade serão tolerados durante o período de atuação do contratado, sendo passíveis de investigação e demissão por justa causa. Colaboradores são encorajados a comunicarem os ocorridos observados a seus *Buddies* ou *Managers*, que por sua vez são os responsáveis por orientar cada recém chegado(a) em como navegar pelas políticas e processos da empresa, além de guiá-los no desenvolvimento de suas carreiras e servirem como porta-vozes dos colaboradores no tratamento de problemas com - ou sugestões para - os líderes da empresa.

O corpo de funcionários do escritório de São Paulo é composto por aproximadamente 40 pessoas, das quais a maioria é de desenvolvedores de software experientes em diversas sub-áreas. A outra parte conta com profissionais de recursos humanos, financeiro, recrutamento e gerência. O logotipo da Avenue Code é apresentado na Figura 2.

Figura 2: Logotipo da Avenue Code.

As suas áreas de atuação de destaque são:

- Consultoria e mentoria de transformação ágil
- UX *Design*
- Desenvolvimento de plataformas
- Microserviços
- Consultoria e implementação de DevOps
- Aplicações móveis nativas
- *Machine Learning* e *Data Science*
- Integrações corporativas

A Avenue Code atende diversos clientes notórios nos mercados brasileiro e internacional. A seguir estão alguns exemplos são:

- FIAT
- Apple
- GAP
- E-Bay
- Fanatics
- SquareTrade

2.1.3 Características da área onde o trabalho foi realizado

O escritório de São Paulo é composto por dois andares (terceiro e oitavo) em um edifício próximo a estação Consolação do metrô de São Paulo. A organização do escritório é guiada pelos princípios do conceito de *Open Office*, ou seja, não existe divisão entre os espaços dos funcionários. Mesas são dispostas pelos andares ao invés de baias, e salas de reuniões com equipamentos de multimídia (televisores, caixas de som e microfones de ambiente) estão disponíveis para quando reuniões com times remotos forem necessárias.

Para reuniões locais e treinamentos, uma parte do oitavo andar possui mesas individuais e cadeiras, dispostas como em uma sala de aula tradicional, em frente a uma parede com uma lousa embutida e um projetor.

À todos os colaboradores é concedido um MacBook de 16GB de memória RAM, 256GB de memória Flash, telas de 13 ou 15 polegadas (dependendo da disponibilidade de computadores no momento da contratação) e processador Core i5 de gerações variadas. Periféricos como monitores, teclados, fones de ouvido, *mouses* e adaptadores diversos também são disponibilizados para solicitação individual.

A infraestrutura conta com uma copa abastecida duas vezes por semana com frutas, café, sucos e refrigerantes, além de uma cozinha equipada com forno microondas, geladeira e pia.

Por fim, uma sala de descompressão também está disponível e conta com um PlayStation 4 com diversos jogos, mesa de pebolim, mesa e raquetes de tênis de mesa e uma decoração com temas de jogos digitais populares.

3 Atividades na Keyrus

3.1 Descrição

O CIP foi a principal atividade em que o colaborador se engajou durante o período compreendido por este relatório. Em especial, com os módulos de ingestão de dados provenientes de redes sociais (Facebook, Twitter, Instagram e YouTube) e da plataforma de análise de dados chamada Crimson Hexagon. A Figura 3 apresenta a tela inicial do CIP.

Figura 3: Tela inicial do CIP.

Além disso, a participação em um curso de metodologia ágil de gerenciamento de projetos, com o professor José Finocchio Junior, criador da metologia Canvas aplicada a gestão de projetos, ocorreu no dia 07/12/2018.

3.2 Ingestão de dados de mídias sociais

A ingestão de dados de mídias sociais do CIP foi implementada utilizando a ferramenta *Software as a Service* (SaaS) de automatização de extração e integração de dados chamada Rivery [7]. A empresa que detém os seus direitos, homônima, é propriedade da Keyrus.

A ferramenta consiste em uma interface de usuário online onde se é possível criar a configurar Rivers. Cada River representa uma integração com alguma fonte de dados, seja uma web API (como as RESTful APIs do Facebook, Twitter, Instagram e YouTube) ou um banco de dados. O

grupo completo de integrações fornecidas pela ferramenta está disponível em seu site oficial. É possível agendar a execução dos Rivers, de forma de todos os Rivers que alimentam o CIP são configurados para gerar novas cargas de dados todos os dias. Também é possível definir o intervalo de dados compreendido por cada execução do River, de forma que a cada execução, o River busca atualizações de dados de dias, meses ou até anos passados.

Cada vez que um River é executado, a carga de dados por ele buscada é inserida no *Data Lake* do CIP como um arquivo no formato JSON, em uma região denominada como *Landing Zone* na arquitetura do *pipeline* de dados planejada para a ingestão. O *Data Lake* é um *cluster* Hadoop, de forma que o arquivo inserido se encontra em seu HDFS (Hadoop *File System*).

O arquivo fica na *Landing Zone* do HDFS até que a execução de uma aplicação de linha de comando chamada datalake_to_sql_server ocorra, portanto, esta execução também precisa ocorrer de forma automática e agendada. No CIP, este módulo foi implantado no *Name Node* do *cluster* (cujo sistema operacional é o Ubuntu 16.04), portanto, para o agendamento de sua execução, é utilizado o crontab. Este módulo é executado pelo crontab quatro vezes ao dia, independente de haverem novos arquivos com dados na *Landing Zone* ou não. No caso de não haverem, a aplicação não faz nada além de gerar um arquivo de *log*, mas se houverem, ela passa a localização de cada arquivo no formato JSON como parâmetro para uma *Stored Procedure* (SP) armazenada no *Data Warehouse* do sistema, que por sua vez é uma instância do Microsoft SQL Server [9]. A *Stored Procedure* que é executada varia de acordo com a origem dos dados que estão sendo ingeridos, de forma que existe uma *Stored Procedure* e um grupo de tabelas de destino diferente para cada mídia social.

A aplicação datalake_to_sql_server foi desenvolvida utilizando a linguagem Python 3 [19]. A escolha da linguagem ocorreu devido a familiaridade dos membros do time com ela, a sua facilidade de aprendizado, riqueza de bibliotecas para as mais diversas funcionalidades, mas principalmente pela biblioteca Pandas [16], que é uma ferramenta sem igual para a tarefa de ler e transformar dados. Ela recebe como parâmetros de linha de comando a origem de dados que deve buscar na *Landing Zone* ('fb' para Facebook, 'ig' para Instagram, 'yt' para YouTube e assim por diante) e o nome da SP que deve ser executada para a ingestão dos arquivos desta origem.

Devido ao *cluster* Hadoop do CIP ser uma instância de um *Platform as a Service da Azure* chamado HDInsight, todo o seu HDFS também é uma instância de outro *Platform as a Service* (PaaS) da Azure chamado Blob Storage, cujo objetivo é armazenar grandes volumes de arquivos. Esta integração entre produtos e serviços da Azure também permite ao SQL Server ler um arquivo diretamente de um *Blob Storage* (e por consequência, de um HDFS), sendo assim, cada SP no *Data Warehouse* do CIP recebe como parâmetro o local de um arquivo JSON no HDFS e efetua a leitura, validação e a atualização / inserção de cada registro em alguma tabela fato pela qual ela é responsável.

Uma vez que a SP foi executada pelo datalake_to_sql_server, se tudo ocorrer bem, o arquivo é movido para outra zona do *Data Lake* chamada *History Zone*, onde ficam todos os arquivos que já foram processados pelo *pipeline* de dados do CIP. Se ele não foi processado corretamente, um *email* automático é enviado para os responsáveis técnicos e o arquivo é movido para uma terceira zona chamada *Reject Zone*, onde ficam todos os arquivos que não puderam ser processados de forma

automática pelo *pipeline*.

O modelo de dados escolhido para todos os grupos de tabelas dentro do *Data Warehouse* foi o Modelo Estrela. Ele foi determinado por sua relativa facilidade de implementação (o que, na prática, significa resultados entregues mais rapidamente), pelo conhecimento prévio dos membros do time e principalmente, por ser suficientemente performático [17] para atender os requisitos do CIP.

Ademais, todas as SPs que inserem / atualizam dados seguem a boa prática de passarem por uma *Staging Area* antes de passarem para a tabela fato, de forma a validar a estrutura, presença de valores nulos não permitidos, quebras de chaves e tipos dos dados antes de qualquer interação com a tabela fato ocorrer. Os dados também são temporalizados por meio de uma coluna na tabela fato que contém a data da última atualização de cada respectiva linha.

O intuito de todo este processo é abastecer o *Data Warehouse* para que os relatórios do CIP consumam-no. Estes relatórios foram desenvolvidos utilizando a ferramenta de BI denominada Qlik Sense. Suas costumizações visuais foram desenvolvidas utilizando, principalmente, Angular 2 e a biblioteca para navegadores do Qlik, em JavaScript.

O CIP conta com relatórios dos mais variados tipos, todos atualizados com o conteúdo atual do banco de dados a cada 12 horas. Os relatórios possuem como objetivo geral monitorar a performance das ações de *marketing* da empresa (gasto por alcance, quanto do alcance é orgânico e quanto é pago, tempo no ar de cada anúncio ou campanha, dentre vários outros) para cada uma de suas marcas, em cada um dos países onde elas atuam, em cada veículo de comunicação distinto e com granularidade mínima de 1 dia. Como ele está em fase de desenvolvimento, nem toda marca e país estão presentes no relatório até novembro de 2018, mas o número de ambos cresce a cada *sprint*. Uma captura de tela do resumo de relatórios do CIP é mostrada na Figura 4.

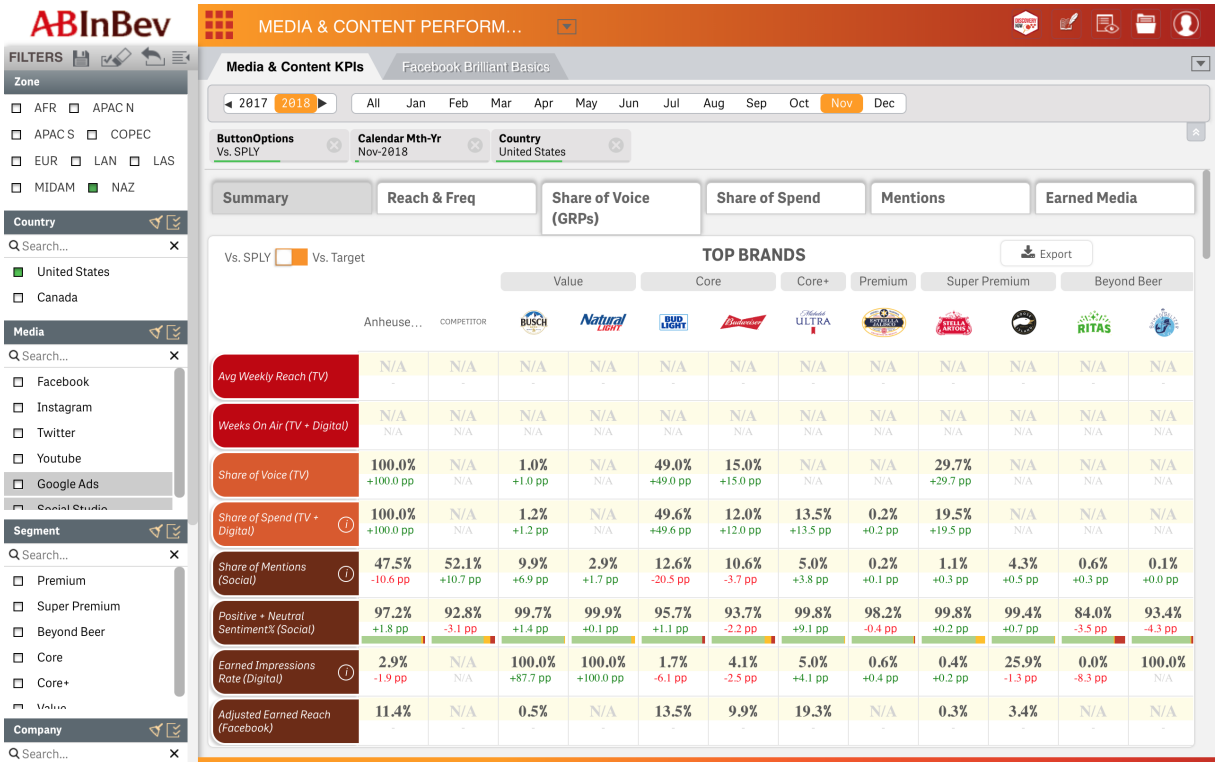

Figura 4: Tela de resumo do CIP.

É possível observar que uma cópia dos dados extraídos das diferentes fontes e inseridos no Data Warehouse fica no *Data Lake*. Isso ocorre para que, no futuro, quando o volume de dados do CIP crescer tanto que um banco de dados relacional comece a apresentar gargalos de performance, exista uma alternativa pré-pronta de acessá-los. Além disso, como consultoria, não se sabe qual o rumo de crescimento que o sistema adotará uma vez que ele seja entregue, e manter os dados no HDFS faz com que seja possível trabalhar sobre eles utilizando as ferramentas associadas ao Hadoop, como Hive, Impala, Sqoop, Spark etc. Estas ferramentas foram pensadas justamente para facilitar o trabalho sobre grandes volumes de dados e amenizam a complexidade de resolução de problemas frequentemente associados a área de *Big Data*, como treinamento de algoritmos de *Machine Learning* (por meio da biblioteca SparkML lib), captura de dados em tempo real (Flume) etc.

3.3 Ingestão de dados do Crimson Hexagon

Além dos dados provenientes diretamente das APIs de redes sociais, o CIP utiliza uma ferramenta à parte para avaliar a percepção do mercado com relação as suas marcas em cada país em que estas atuam. Esta ferramenta se chama Crimson Hexagon [1].

Crimson Hexagon consiste em um SaaS com uma UI online onde se é possível criar e configurar monitores. Cada monitor possui um conjunto de palavras-chave (e uma linguagem simplificada para criação de restrições e associações de palavras, delimitadores de escopo etc), um grupo de mídias sociais, um intervalo aberto ou fechado de tempo e uma delimitação geográfica. Tirando as palavras-

chave, todos os outros parâmetros são opcionais e caso não sejam fornecidos, o escopo mais geral possível é assumido (por exemplo, se nenhuma restrição geográfica for configurada, a busca e análise de dados ocorrerá para dados capturados de usuários do mundo todo).

Uma vez que um monitor esteja criado, um identificador único é atribuído a ele e *web scrappers* buscam dados nas mídias sociais selecionadas. Após isso, métricas são computadas para este monitor específico, como quantidade de *posts* (com intervalo de tempo ajustável), quantidade de impressões, análise de sentimento dos *posts*, entre várias outras.

Os gráficos destas métricas podem ser visualizados ao acessar o monitor pela UI, mas elas também podem ser extraídas da RESTful API [2] que acompanha o produto, e é precisamente isso que o módulo crimson_to_datalake, também desenvolvido em Python 3, faz.

crimson_to_datalake é uma aplicação de linha de comando cuja finalidade é enviar repetidas requisições à *web* API do Crimson Hexagon, agregar os resultados, salva-los em um arquivo JSON e mover este arquivo resultante para a supracitada *Landing Zone*.

Ela recebe como parâmetro apenas o tipo de métrica que o usuário deseja buscar (pode ser 'results' ou 'top_sources'). Este parâmetro define o *endpoint* da API para qual as requisições serão enviadas, e portanto, o tipo de métrica que estará contido na resposta, que serão análises de sentimento de *posts* ou informações de volumetria de *posts* por veículo, respectivamente.

Cada requisição, independente do *endpoint*, requer os mesmos parâmetros: as datas de início e fim do intervalo de interesse e o identificador do monitor que contém as métricas de interesse. Estes parâmetros são enviados via URL.

A lista de identificadores de monitores associados ao CIP é contida em uma tabela de dimensão do *Data Warehouse*, portanto uma consulta a esta tabela é feita antes de iniciar o *loop* de requisições a API do Crimson Hexagon.

Uma vez que a aplicação seja executada e os dados extraídos estejam na *Landing Zone*, basta que o módulo explicado anteriormente, o datalake_to_sql_server, seja executado (obviamente, recebendo os parâmetros que o instruam a tratar dos arquivos do provenientes do Crimson Hexagon) para que estes dados sejam inseridos / atualizados no *Data Warehouse* e os arquivos sejam movidos para a *History* ou *Reject Zone* do *Data Lake*.

A construção da aplicação crimson_to_datalake foi necessária pois o Rivery não possui integração com a API do Crimson Hexagon até o presente momento. Além disso, o modelo de dados no *Data Warehouse* dos dados provenientes do Crimson Hexagon também é o Estrela, pelas mesmas razões citadas anteriormente.

3.4 Curso de projetos ágeis

O curso teve duração de um dia, ocorreu em 07/12/2018 e foi ministrado pelo professor José Finocchio Junior [4], criador da metodologia Canvas de gerenciamento de projetos.

Nele, o básico de como projetos ágeis devem ser conduzidos foi abordado enquanto os alunos propunham suas respectivas ideias por meio do preenchimento de um Canvas.

Primeiro, foi solicitado que cada grupo de seis alunos pensasse em um novo projeto para a Anheuser-Busch InBev e apresentassem esse projeto para que os outros grupos avaliassem e votassem pela destinação de recursos financeiros, similar ao que ocorre no programa de TV denominado *Shark Tank*. O projeto apresentado pelo meu grupo foi uma Rede Neural Artificial cujo objetivo seria identificar grupos (étnicos, faixa etária e/ou geolocalidades, por exemplo) mal atendidos pela AB InBev ao ser treinada utilizando o cruzamento de dados de mídias sociais já capturados pelo CIP com dados de vendas.

Uma vez que essa fase inicial, chamada de *pitching*, foi concluída, os estudantes de cada grupo foram, a cada nova explicação, preenchendo cada espaço em branco no Canvas com informações como o conjunto de *Stakeholders*, a razão não técnica do projeto ser necessário, quais as premissas para que ele seja implementado, quais profissionais e quantos de cada serão precisos, qual o custo, quais as funcionalidades que os usuários podem esperar do projeto quando concluído etc. O Canvas que foi preenchido encontra-se na Figura 5. A finalidade de se preencher o Canvas é obter respostas para seis questões fundamentais sobre o projeto: por quê construí-lo? O que construir? Quem são as partes interessadas? Como construí-lo? Quando será entregue? Quanto custará?

Quando o Canvas foi finalizado, os alunos foram instruídos a dividir as funcionalidades em Estórias de Usuário, que são descrições dos requisitos no formato "Como um _____, eu desejo que _____, com o objetivo de _____", em que cada espaço em branco representa um tipo de usuário, uma funcionalidade descrita superficialmente e qual o objetivo desta funcionalidade existir, respectivamente.

Conforme o dia foi se passando, estratégias de como priorizar Estórias de Usuário, como distribuí-las pelos membros do time, quais ferramentas utilizar para cada metologia (SCRUM e Kanban, por exemplo) o que fazer em caso de atraso de alguma entrega de usuário, como usar a cerimônia do *Planning Poker* para estimar esforços de cada tarefa e como lidar com diversos outros problemas que frequentemente surgem em um projeto ágil foram discutidos.

Por fim, o professor explicou que é mais produtivo que cada profissional do time esteja alocado para desenvolver um número pequeno de tarefas (idealmente, apenas uma, mas nunca ultrapassando três) ao mesmo tempo, e fez com que os estudantes testassem essa afirmação por meio da construção conjunta de pequenos bonecos de LEGO.

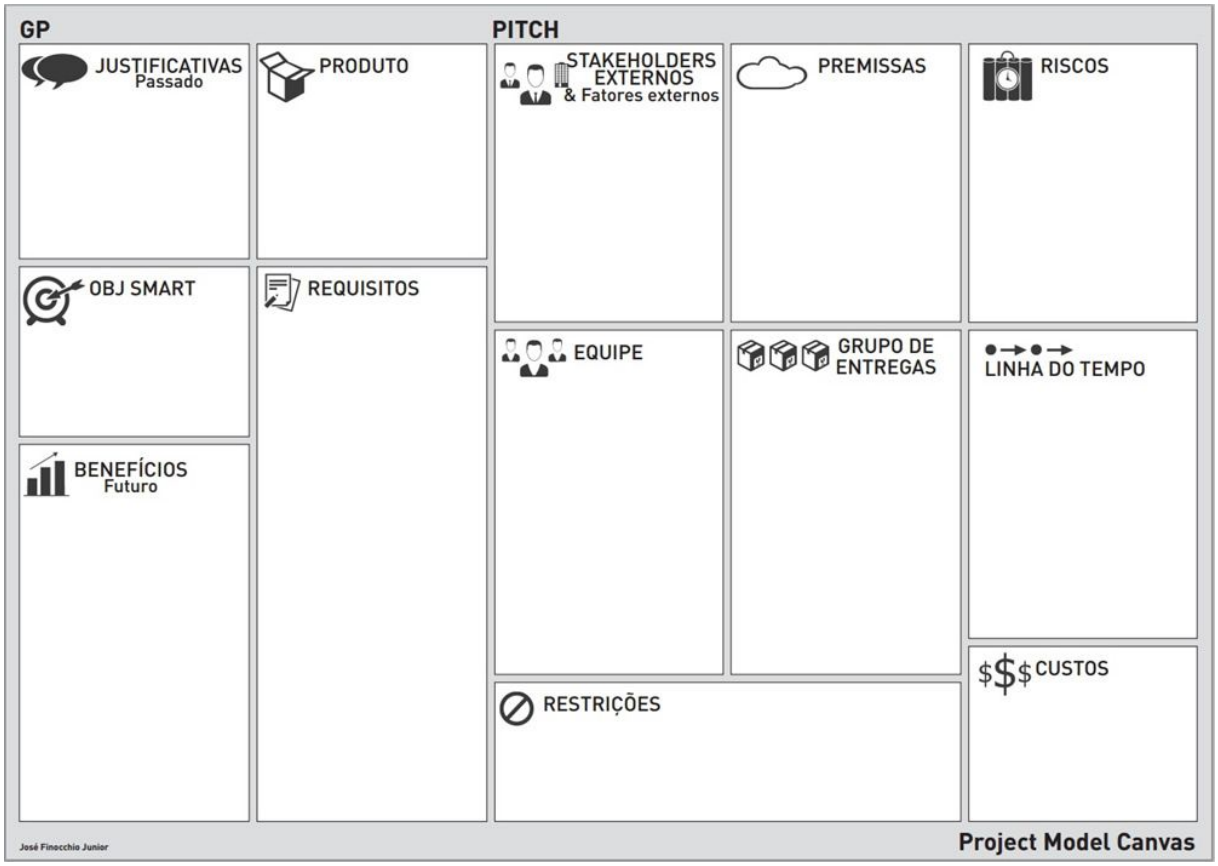

Figura 5: Canvas em branco.

4 Atividades na Avenue Code

4.1 Atividades desenvolvidas durante o Estágio Supervisionado II

4.1.1 Descrição

Durante as primeiras semanas de atuação na Avenue Code ocorreram uma consultoria pontual para um possível futuro cliente no Brasil - a Samsung - e uma palestra sobre os aspectos técnicos da metodologia SCRUM aplicada a projetos de software - quem são os profissionais envolvidos no time, quais os seus respectivos escopos de atuação, quais as cerimônias requeridas, suas periodicidades ideais e motivações. Contudo, a atividade mais duradoura em que o colaborador se engajou foi o estudo da quarta versão da ferramenta Mule ESB. Isto ocorreu devido ao colaborador possuir familiaridade prévia com a versão 3, mas demonstrar um déficit em suas habilidades relacionadas a versão mais recente, que representa um espaço no mercado de consultorias de TI onde a Avenue Code pretende se encaixar. Sendo assim, os responsáveis técnicos concluíram que esta seria a melhor forma de investir o tempo de um recém-contratado que ainda não havia sido alocado em nenhuma equipe para atender às necessidades de um cliente fixo.

4.1.2 Consultoria pontual à Samsung

A primeira atividade em que o colaborador se engajou foi uma consultoria pontual prestada à Samsung Brasil.

A empresa cliente possuía um sítio eletrônico, desenvolvido por outra consultoria, cujo objetivo era a apresentação de produtos Samsung promocionais, comercializados por diversos de seus parceiros varejistas, durante uma campanha de promoções organizada por ela, denominada UpHouse, e assim direcionar possíveis compradores para os respectivos *e-commerces* de seus parceiros após o eles(as) manifestarem interesse de compra. Em outras palavras, o sítio eletrônico em questão desempenhava o papel de funil de vendas da campanha de promoções.

O problema apresentado aos representantes da Avenue Code foi o seguinte: após alguns minutos online e alguns milhares de requisições HTTP respondidas pela aplicação, o sítio eletrônico parava de respondê-las definitivamente, até que alguém acessasse o servidor e reiniciasse o seu processo manualmente.

O código-fonte do sítio eletrônico foi concedido aos profissionais técnicos da Avenue Code, sob assinatura de um termo de sigilo, assim como acesso ao servidor de aplicações responsável por mantê-lo *online*.

Após a execução de um teste de carga enquanto avaliando o comportamento da aplicação no servidor, a equipe constatou que o montante de memória RAM do servidor de aplicações consumido pelo sítio crescia demais após algumas centenas de requisições, o que fazia com que o sistema operacional abortasse seu processo.

Em seguida desta conclusão, o código-fonte foi avaliado por algumas horas pelo time técnico. Ele foi escrito utilizando um *framework* proprietário da Microsoft denominado ASP.NET Razor. A equipe técnica, por sua vez, constatou que a conexão com o banco de dados responsável por armazenar a lista de produtos promocionais não estava sendo fechada após o uso, de forma que a mesma continuava a consumir memória RAM do computador mesmo após a consulta ocorrer e o resultado ser enviado ao usuário. A longo prazo, este vazamento de memória era responsável por provocar o interrompimento do funcionamento do sítio eletrônico.

A consultoria responsável pelo desenvolvimento foi comunicada da falha, mas devido a urgência requerida pelo cliente para que o sítio eletrônico ficasse disponível e operante, os atributos consultados do banco de dados foram copiados e estruturados dentro um arquivo de dados no formato JSON, e o código-fonte do sítio foi refatorado para passar a consultar seus valores deste arquivo ao invés de abrir uma conexão com um banco de dados remoto. Esta solução foi paliativa e não é recomendada para aplicações desenvolvidas de forma planejada.

Um relatório que detalhava todo o processo de *troubleshooting* executado pelos profissionais técnicos da Avenue Code, bem como a conclusão sobre a falha que causava o problema e a solução paliativa implementada, foi escrito e encaminhado para os responsáveis pela campanha na Samsung.

4.1.3 Curso online MuleSoft.U Mule 4 for Mule 3 Users

O conhecimento descrito na primeira subseção foi adquirido pelo colaborador por meio de estudo individual durante o período de atividades para que o curso - descrito na segunda subseção - fizesse sentido.

Introdução ao Mule

Mule é uma PaaS cujo principal objetivo é facilitar a construção de camadas de integração entre sistemas corporativos [15]. O ambiente de execução dos programas (também chamado de *Mule Runtime*) foi construído sobre o *framework* Spring [8], que por sua vez é baseado na linguagem Java [13] e é executado sobre a *Java Virtual Machine* (JVM), logo, todo programa desenvolvido na plataforma Mule é, por definição, um programa em Java. Isto fica claro ao observar que o arquivo resultante do processo de compilação de uma aplicação desenvolvida sobre Mule versão 4 possui o formato JAR.

As principais formas pelas quais a plataforma simplifica a construção de integrações entre sistemas são prover:

- um ambiente de desenvolvimento visual (chamado Anypoint Studio) com componentes pré construídos e altamente configuráveis;
- um repositório de componentes, chamado Exchange, onde usuários podem compartilhar seus componentes e utilizar os componentes compartilhados por outros. Componentes podem ter os mais variados comportamentos, desde a aplicações de uma transformação de dados comum até a facilitação do consumo de uma RESTful API popular;
- um ambiente de computação em nuvem, denominado CloudHub, que se integra facilmente com o Anypoint Studio e facilita o processo de implantação de novas versões de projetos;
- uma documentação navegável e implementação preliminar de RESTful APIs, inclusive com validação de erros, recebendo como parâmetro apenas a definição do contrato da API no formato RAML, por meio do componente APIKit;
- a DataWeave [3], uma linguagem de *scripting* cuja principal aplicação é a transformação de dados em memória.

Além disso, qualquer comportamento desejado por algum(a) desenvolvedor(a) que não possa ser atingido por meio da utilização de um componente, pode ser atingido por meio da escrita de *scripts* em Java, Python, Ruby ou Groovy. Obviamente, Java e Groovy mostram maior desempenho e possuem suporte mais avançado devido a serem naturalmente mais próximas do ecossistema de tecnologias baseadas na JVM.

Cada aplicação Mule é baseada em fluxos (ou *flows*) que são disparados por algum tipo de gatilho configurável, por exemplos: o sinal de um agendador cronológico, o recebimento de uma requisição HTTP, a chegada de uma nova mensagem em uma fila de mensagens etc. Os fluxos são definidos

por arquivos de configuração no formato XML. O Anypoint Studio renderiza o código XML (que será traduzido para Java no momento da compilação) na forma visual apresentada na Figura 6.

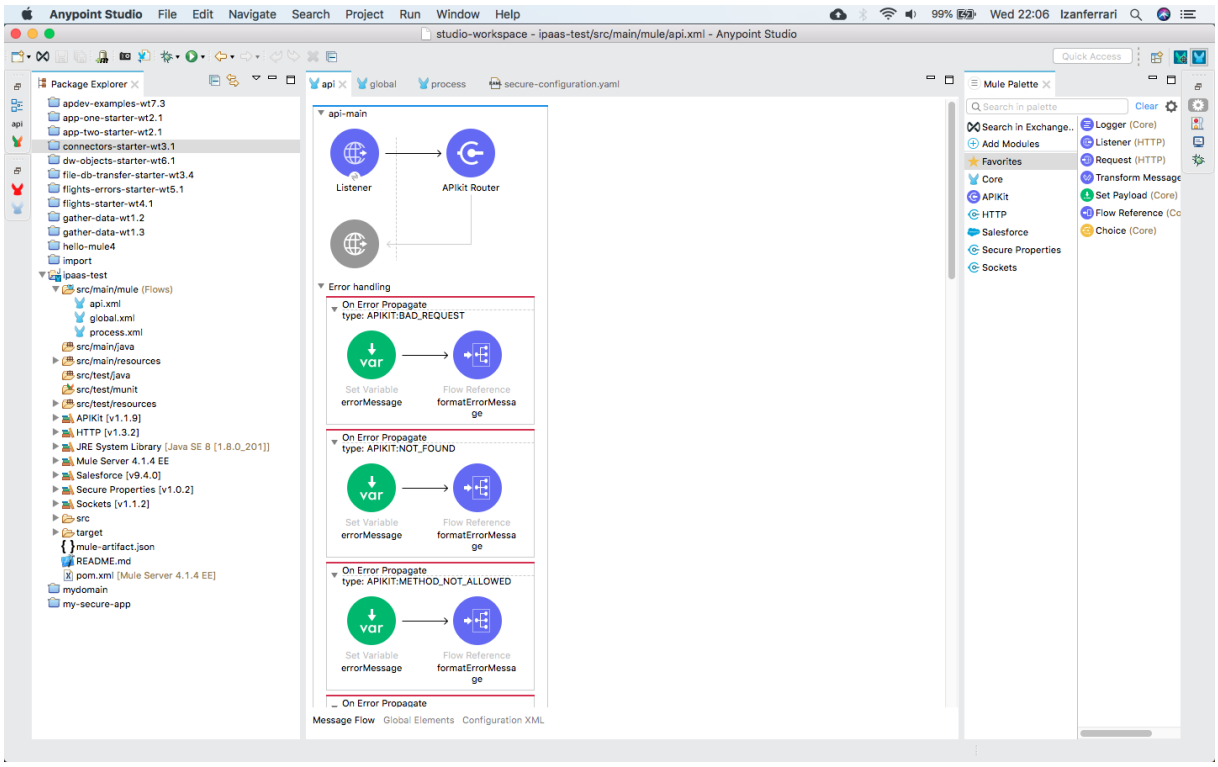

Figura 6: Anypoint Studio com o arquivo api.xml aberto.

Por fim, a utilização da plataforma em si já representa ganhos em velocidade de entregas e simplificação da complexidade durante a construção de um ESB, mas a forma mais eficiente de se tirar proveito dela é por meio da estruturação do ESB sob o padrão arquitetural denominado *API-Led Connectivity* [11].

Este padrão arquitetural prega a necessidade de estruturar sua rede de aplicações (ou APIs) em 3 camadas: de sistema, de processamento e de experiência. As aplicações clientes apenas interagem com as APIs da camada de experiência, que por sua vez interage apenas com as APIs da camada de processamento, enquanto as APIs desta camada interagem apenas com as da camada de sistema.

A camada de sistema visa expor os sistemas a serem integrados por meio de uma interface comum (normalmente, uma RESTful API para cada sistema). A camada de processamento visa integrar, limpar, validar e, resumidamente, aplicar qualquer lógica de negócio sobre os dados expostos pelas aplicações da camada de sistemas, e expor o resultado disto por meio de uma interface comum também. Por fim, as aplicações da camada de experiência devem ser responsáveis por simplificarem a experiência dos consumidores da rede de aplicações, integrando entidades virtuais da camada de processamento, reformatando respostas de requisições, validando erros, etc. A Figura 7 esquematiza o que foi explicado por meio de um exemplo.

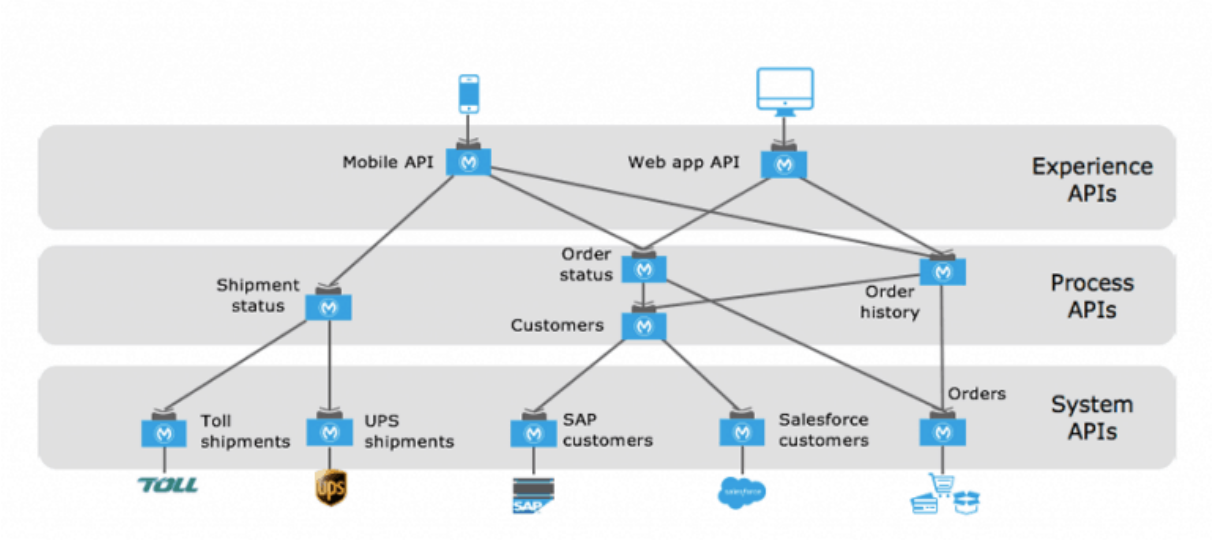

Figura 7: Diagrama que exemplifica a aplicação do API-Led Connectivity.

O benefício deste padrão mora no desacoplamento das APIs. É possível observar que se uma alteração na lógica de integração for necessária, ou algum sistema ficar obsoleto, apenas as APIs da camada processamento e/ou de sistema precisam ser modificadas, enquanto a camada de experiência pode se manter inalterada e por consequência, as aplicações que as consomem não precisam sentir o impacto da mudança. Além disso, o reuso de APIs é facilitado se, por exemplo, uma nova camada de experiência se tornar necessária.

É importante salientar que este modelo de arquitetura não é dependente da plataforma Mule, mas é significativamente facilitado por ele devido a possibilidade de se expor APIs por meio da publicação de conectores no Exchange.

Conteúdo do curso

O curso, cuja página de registro - com resumo e descrição - se encontra em [5], intenciona apresentar as funcionalidades do Mule 4 para desenvolvedores com experiência prévia com Mule 3. Ele é dividido em sete módulos e seus pré-requisitos incluem não apenas conhecimento prévio na plataforma Mule 3, mas também em uma linguagem de programação orientada a objetos, com preferência para Java por razões triviais, além de conhecimentos sobre comunicação via internet utilizando o protocolo HTTP, serviços de fila de mensagens, bancos de dados etc.

Cada módulo tratou de um escopo de atividades que desenvolvedores Mule precisam atuar sobre com frequência, sempre mencionando como os problemas deste escopo de atividades são resolvidos na versão nova da plataforma em detrimento de como eram ao utilizar-se a versão antiga.

A lista de módulos e os seus tópicos se encontra a seguir.

- Módulo 1: apresenta como a estrutura de pastas inicial de aplicações Mule 4 foi alterada, além de descrever mudanças no formato do objeto de mensagem que trafega pelos *flows* e comenta

sobre a nova forma de ajuste de performance que a plataforma oferece. Todo fluxo é agora, por padrão, não bloqueante e assíncrono. Além disso, menciona que na nova versão, toda aplicação Mule tem suas dependências gerenciadas pelo Maven por padrão;

- Módulo 2: mostra como configurar e utilizar um arquivo de propriedades hierárquico no formato YAML, uma novidade do Mule 4, além de criptografá-las com o novo componente de propriedades seguras. Também apresenta como criar e configurar uma aplicação de Domínio (ou aplicação pai) para outras aplicações;
- Módulo 3: apresenta a nova forma de estruturação de componentes. Antes, componentes representavam um serviço - por exemplo, havia um componente *File* que encapsulava todas as operações sobre arquivos que a plataforma disponibilizava. A partir da nova versão, cada operação de está encapsulada em um componente diferente. A seguir, o módulo aborda como configurar a utilizar conexões com sistemas de mensageria, *file streams* e como importar bibliotecas em Java para seu projeto;
- Módulo 4: trata de como escrever testes unitários e de integração utilizando o MUnit, um *framework* baseado no JUnit cujo objetivo é testar fluxos. Também apresenta o conector Choice, que serve para implementar estruturas condicionais;
- Módulo 5: conectores e tipos de fluxo especiais para o tratamento de exceções são introduzidos;
- Módulo 6: a nova versão da linguagem DataWeave é abordada de forma prática, assim como os novos componentes que devem ser utilizados para a implementação de transformações de dados personalizadas baseadas em classes Java;
- Módulo 7: conectores de gatilho são abordados e seus diferenças para com os conectores disponíveis na versão anterior da plataforma, assim como fluxos do tipo batch (destinados ao processamento assíncrono de volumes de dados maiores).

Ademais, o curso conta com um módulo opcional extra repleto de exercícios para fazer sem auxílio. O intuito destes exercícios é capacitar o(a) estudante para as provas de certificação *MuleSoft Certified Developer - Level 1 (Mule 4)*.

4.1.4 Palestra sobre a metologia SCRUM

A palestra foi ministrada por Bruna Vaz, uma das *SCRUM Masters* mais experientes da equipe da Avenue Code. Ela, profunda conhecedora do conteúdo solicitado nas provas de certificação *Professional Scrum Master* (PSM) I e II, utilizou-o como guia para a elaboração de sua apresentação e baseou-se no livro *SCRUM: The Art of Doing Twice the Work in Half the Time* [18].

Primeiro, foram apresentados os diferentes papéis que compõe um time de acordo com os moldes do SCRUM. São eles: *Product Owner*, *SCRUM Master*, time de desenvolvimento (cuja composição varia de acordo com o projeto a ser entregue a cabe ao seu responsável, geralmente um arquiteto de soluções, definir quais os tipos de profissionais e quantos de cada serão necessários). Além disso,

foi mencionado que, de acordo com sua experiência, a figura de um gerente de projetos costuma ser benéfica ao desenvolvimento do projeto, apesar de não ser prevista pela literatura, pois retira do *Product Owner* e do *SCRUM Master* parte das obrigações burocráticas desempenhadas por posições de gerência e permite que ambos foquem seus esforços na manutenção da metodologia conforme o projeto avança.

Cada uma das funções foi explicada, e todas serão reproduzidas abaixo.

- *Product Owner* (PO): é o responsável por captar dados sobre as expectativas dos *stakeholders*, frequentemente por meio de entrevistas, e definir as estórias de usuário que comporão o projeto. Estas estórias costumam seguir o formato "eu, como _____, quero _____, pois _____", sendo o primero espaço em branco um tipo de usuário, o segundo, a descrição macro de uma funcionalidade que este usuário espera que o produto possua, e o terceiro, a justificativa para o desejo de tal funcionalidade na visão do usuário. Também é responsável por acompanhar o que está sendo desenvolvido pelo time técnico de perto e validar se o progresso realmente atende as expectativas a cada iteração;
- *SCRUM Master* (SM): coordena e auxilia o time como um solucionador de problemas de escopo geral, tirando dúvidas e desobstruindo caminhos, além de garantir que a metodologia está sendo seguida da forma correta, pois apenas assim os seus benefícios podem ser observados. É o responsável também por organizar as cerimônias do SCRUM, como por exemplos: reuniões diárias em pé, *planning poker* e revisão das *sprints*;
- Time de desenvolvimento: UX *designers*, engenheiros de software, desenvolvedores de *Quality Assurance*, devOps, testadores, engenheiros de aprendizado de máquina, cientistas de dados, arquitetos de soluções, analistas de negócios etc. Em suma, qualquer profissional que exerça uma função de caráter técnico e contribua para a construção do produto de forma direta.

Após isso, o conceito de *sprint* foi definido - período de duas semanas (que pode ser variado se necessário, apesar disso ser desencorajado) durante o qual os membros do time de desenvolvimento trabalham para fazer com que as estórias de usuário priorizadas pelo(a) PO sejam implementadas e componham o produto final - e as principais cerimônias foram apresentadas. São elas:

- Reuniões diárias em pé: reuniões de aproximadamente 15 minutos onde cada membro do time responde, individualmente, as seguintes três questões em voz alta: o que você fez ontem? O que você está fazendo (ou pretende fazer) hoje? Você possui algum impeditivo para dar prosseguimento em seu trabalho?
- *Planning poker*: reunião no começo de cada *sprint* em que cada membro do time atribui de forma secreta uma carta que representa um valor a uma estória de usuário. Este valor deve representar o nível de dificuldade que o profissional espera que a estória represente. Esta cerimônia é importante pois assim o PO e o SM são capazes de captar dados e mensurar a performance do time (ao comparar a média do valor atribuído pelo time a uma estória com

o tempo que ela levou para ser concluída de fato), permitindo assim a melhor alocação de recursos no decorrer do projeto;

- Revisão das *sprints*: o time deve se reunir no final de cada *sprint* para conversar de forma franca com o PO e o SM, expôr problemas observados que poderiam ser evitados e como evitá-los, fazer críticas e sugestões, além de apontar os acertos observados para que estes possam ser reproduzidos no futuro.

Por fim, foi possível observar que o papel do *SCRUM Master* se torna obsoleto após o time amadurecer suficientemente sua compreensão do SCRUM e capacidade de trabalhar em equipe com determinada configuração, portanto, na rara ocasião de um determinado time permanecer unido tempo suficiente para atingir este nível de maturidade, a indicação oficial é que este a pessoa que exerce este papel seja realocada para outro time onde seja necessária.

4.2 Atividades desenvolvidas durante o Estágio Supervisionado III

4.2.1 Descrição

Durante o período compreendido pelo Estágio Supervisionado III, o profissional foi designado para atuar como consultor técnico e desenvolvedor de uma série de aplicações demonstrativas construídas sobre a plataforma da MuleSoft. O intuito destas demonstrações foi compor o arsenal de recursos tecnológicos que poderiam ser utilizados pelo cliente, o *Chief Technology Officer* da própria MuleSoft, em uma série de apresentações que ele fará em uma conferência global, com estréia em Atlanta, chamada MuleSoft Connect 2019. Estas apresentações receberam o nome de *Future of Service* (Futuro do Serviço), pois a narrativa fictícia ocorre em 2025 e é baseada em uma família planejando e conduzindo uma viagem de final de semana em conjunto, e resolvendo os problemas que advém deste contexto por meio de plataformas de serviços digitais.

4.2.2 Carro inteligente

No primeiro capítulo do enredo, a família está em um carro alugado e, sem conhecimento, experimenta problemas de vazamento de ar no pneu traseiro esquerdo. O problema é identificado pela companhia locadora, que mantém sensores de velocidade de rotação e pressão nos pneus de seus carros, e assim notifica a família e o substitui em um ponto de encontro.

Para demonstrar como este serviço poderia funcionar no mundo real, uma rede de aplicações foi desenvolvida. A primeira foi uma RESTful API que recebia objetos JSON - provenientes das requisições HTTP disparadas pelos sensores - e os acumulava por alguns segundos em memória. Após o intervalo, a aplicação publica as listas de leituras acumuladas em duas filas (criadas utilizando o AnypointMQ) - uma para leituras de pressão e outra para as de velocidade de rotação do pneu. A segunda foi uma aplicação Mule que consumia estas leituras e as direcionava para uma aplicação gráfica hospedada um serviço da SalesForce chamado Lightning por meio do disparo de eventos de

plataforma. A terceira e quarta aplicações desta rede foram, intuitivamente, a aplicação gráfica - cujo objetivo era exibir as leituras recebidas em formato de gráfico - e o conjunto de hardware (o modelo físico de um carro, um SuperDroid Robots IG42 SB de 4WD de tração com pneus de 10 polegas [10], com uma Raspberry Pi acoplada) e software (programa para capturar leituras dos sensores e envia-las via requisições HTTP) que representaria o carro inteligente.

A aplicação gráfica pode ser observada na Figura 8.

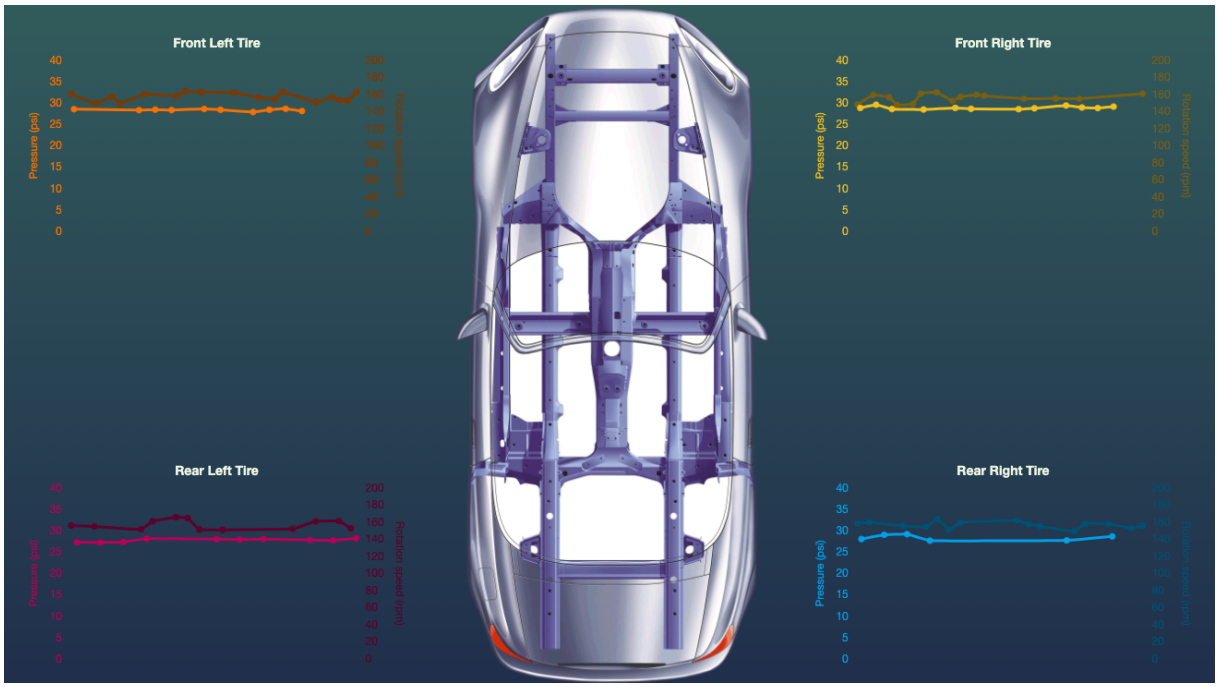

Figura 8: Aplicação gráfica da demonstração de carro inteligente.

Os principais fluxos da implementação da API - aqueles que implementam a lógica de negócio - e da aplicação que publica os eventos de plataforma seguem nas Figuras 9 e 10

Figura 9: Implementação da API que recebe as leituras dos sensores do carro.

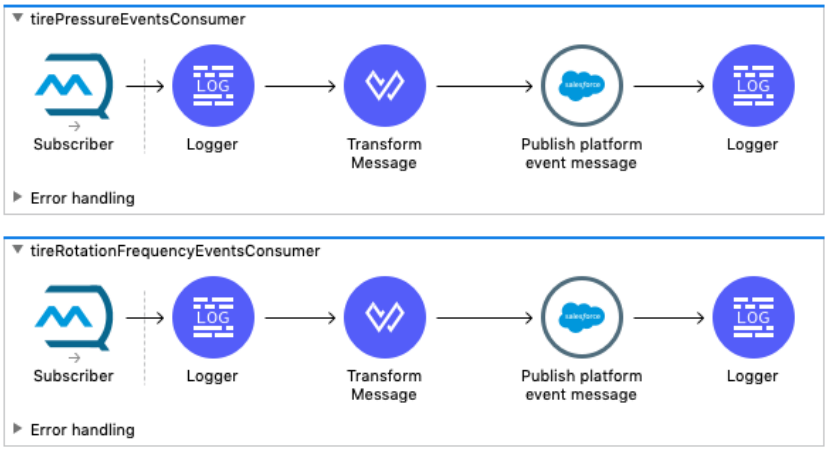

Figura 10: Implementação da aplicação publica as leituras dos sensores como eventos de plataforma.

Para o desenvolvimento da RESTful API, um componente Mule chamado APIKit foi utilizado. Este componente lê uma especificação de API escrita em alguma linguagem (atualmente o suporte mais robusto é oferecido para a linguagem RAML, mas OpenAPI e JSON Schema também são suportados) e gera fluxos para responder a cada rota definida na especificação, além de fazer validação das estruturas de entrada e saída de dados, definir os códigos HTTP de cada resposta possível, estratégias de tratamentos de erro e formas de segurança da implementação da API. Por fim, ele também gera um console interativo para que as APIs desenvolvidas com base nele possam ser testadas sem o auxílio de ferramentas externas uma vez que estejam hospedada em algum computador - mesmo que este seja um ambiente de desenvolvimento pessoal. A Figura 11 mostra o console da RESTful API construída para receber as leituras provenientes dos sensores do carro, gerado pelo APIKit. A especificação desta API, escrita em RAML, encontra-se no apêndice.

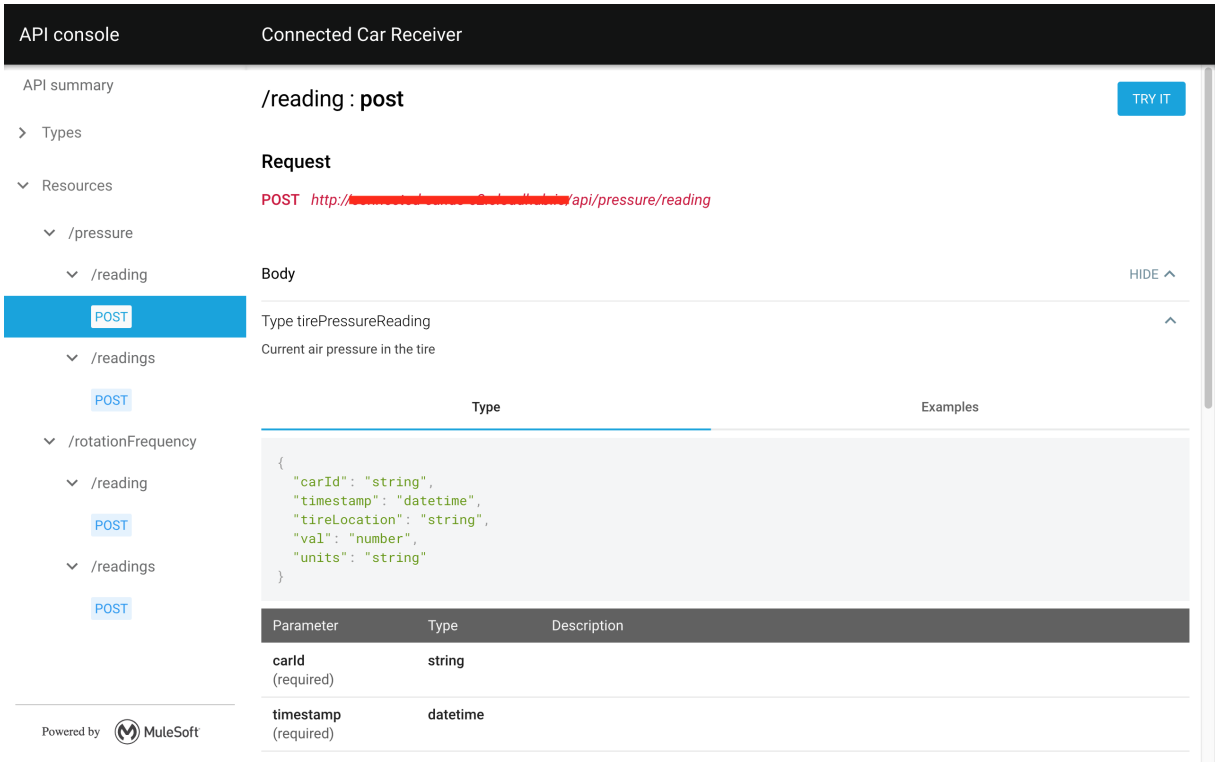

Figura 11: Console de API gerado pelo APIKit.

4.2.3 Negociação automatizada em marketplaces

No segundo capítulo do estória, a família está dentro do novo carro, mas presa no trânsito e o humor no ambiente está em declínio. O aplicativo de GPS informa que a chegada ao destino da viagem será atrasada em diversas horas, de forma que o final de semana em família passa a correr risco de se tornar um fracasso. A assistente pessoal da família - uma rede de aplicações presente em todos os seus dispositivos eletrônicos, inclusive o celular - identifica de forma autônoma esta alteração de humor no ambiente através de leituras das expressões faciais de todos, assim como também identifica o risco à viagem e assim passa a atuar em prol da família para procurar uma alternativa para os planos iniciais e assim salvar o final de semana. De acordo com a narrativa, no futuro todo serviço possuirá uma API própria que outras aplicações poderão consultar e isso tornará viável a existência de espaços de negociação (ou *marketplaces*) completamente automatizados. Dado que a assistente pessoal da família conhece suas preferências, orçamento, tempo disponível e objetivos, ela passa a enviar mensagens (ou eventos) para certos *marketplaces*, que por vezes ignoram-na ou retornam ofertas para ela. Por fim, a assistente seleciona a melhor oferta e a apresenta a família, que de bom grado aceita a mudança de planos.

A Figura 12 consiste em uma captura de tela da aplicação visual desenvolvida para demonstrar como a assistente pessoal da família é capaz de identificar os sentimentos das pessoas através da avaliação de suas faces. É possível observar pelos objetos JSON que surgem no painel a direita que, conforme a condição do tráfego de carros piora (e por consequência, a emoção capturada também), alternativas

locais de substituição para os planos iniciais da família começam a surgir.

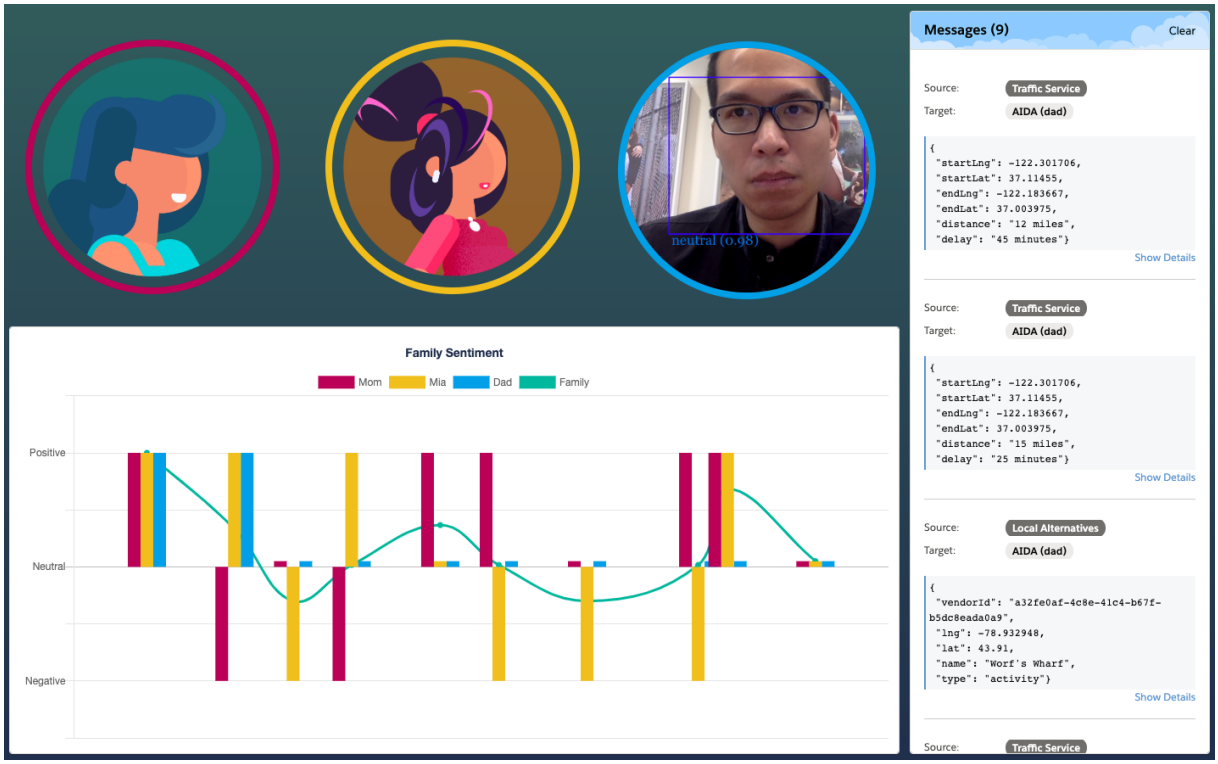

Figura 12: Aplicação de exemplo de avaliação de emoções via expressões faciais em tempo real.

A arquitetura da rede de aplicações definida para demonstrar como ocorreria a interação com o espaço de negociação se encontra na Figura 13. Desta vez, o mecanismo de mensageria escolhido foi o Apache Kafka, apenas para mostrar à plateia que a plataforma é versátil e não está intrinsecamente acoplada a nenhuma ferramenta específica.

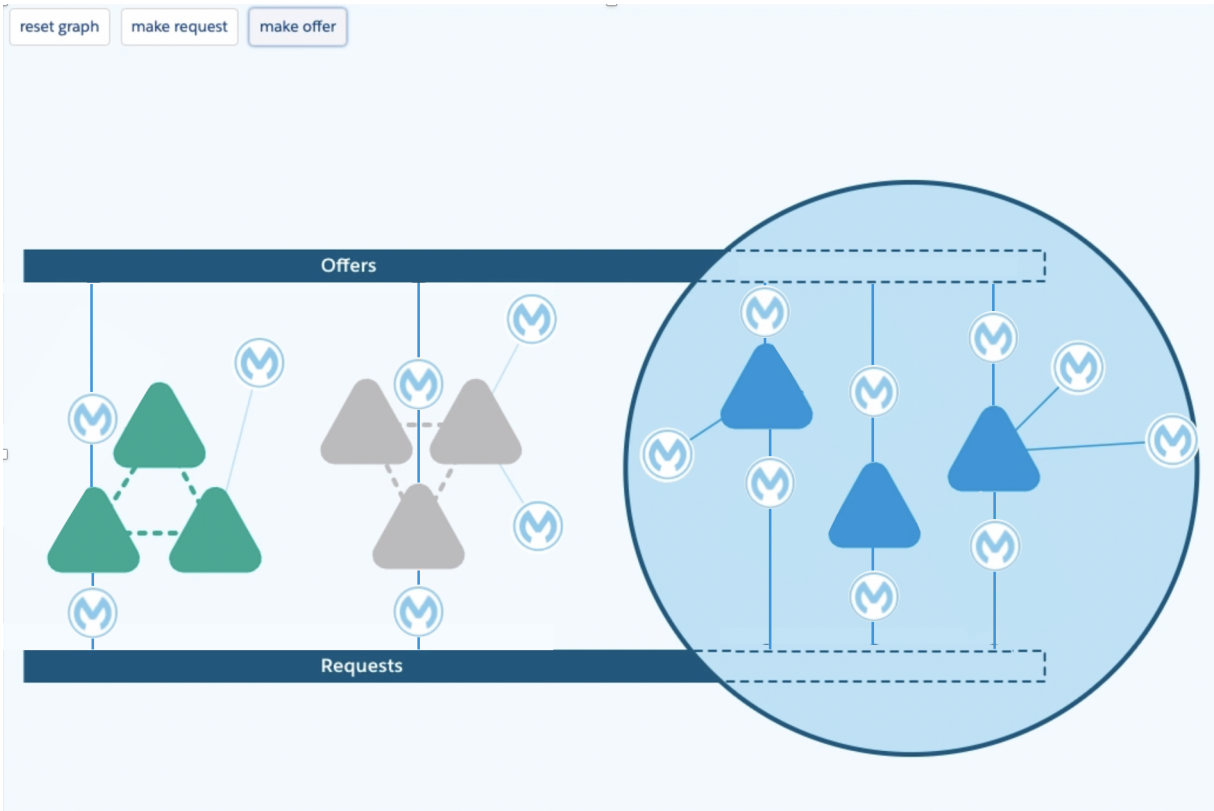

Figura 13: Arquitetura da negociação entre aplicações.

Apesar de a interface com o *marketplace* ser uma RESTful API (aplicação Mule entre o grupo de triângulos verdes - que representa a assistente pessoal da família - e o tópico *Requests*), o seu modo de operação interno é baseado em eventos assíncronos que trafegam pelos dois tópicos presentes na imagem (retângulos azuis escuros denominados *Requests* e *Offers*). O primeiro tópico recebe eventos de pedidos de ofertas, enquanto o segundo recebe as ofertas após as aplicações do *marketplace* (triângulos azuis dentro do círculo) consultarem as APIs de seus fornecedores e computarem uma oferta. A última aplicação da arquitetura (aplicação Mule entre o tópico *Offers* e a assistente pessoal) recebe as ofertas, acumula-as em uma lista e envia-as de volta para a assistente pessoal da família. A assistente pessoal em cinza está presente apenas para demonstrar que mais de uma família pode submeter requisições ao espaço de negociação ao mesmo tempo.

Os tópicos do Kafka foram provisionados utilizando o serviço do Heroku que o engloba. As outras aplicações da rede foram implantadas no CloudHub, assim como todos os recursos tecnológicos baseados em Mule descritos neste texto.

A Figura 14 apresenta a implementação da RESTful API de entrada. A sua definição parcial em RAML (sem as definições dos tipos de dados) pode ser encontrada no apêndice. As definições dos tipos de dados das especificações em RAML foram ocultados para respeitar o acordo de não divulgação que foi feito com o cliente. A aplicação Mule responsável por acumular as ofertas e enviá-las à assistente pessoal se encontra na Figura 15.

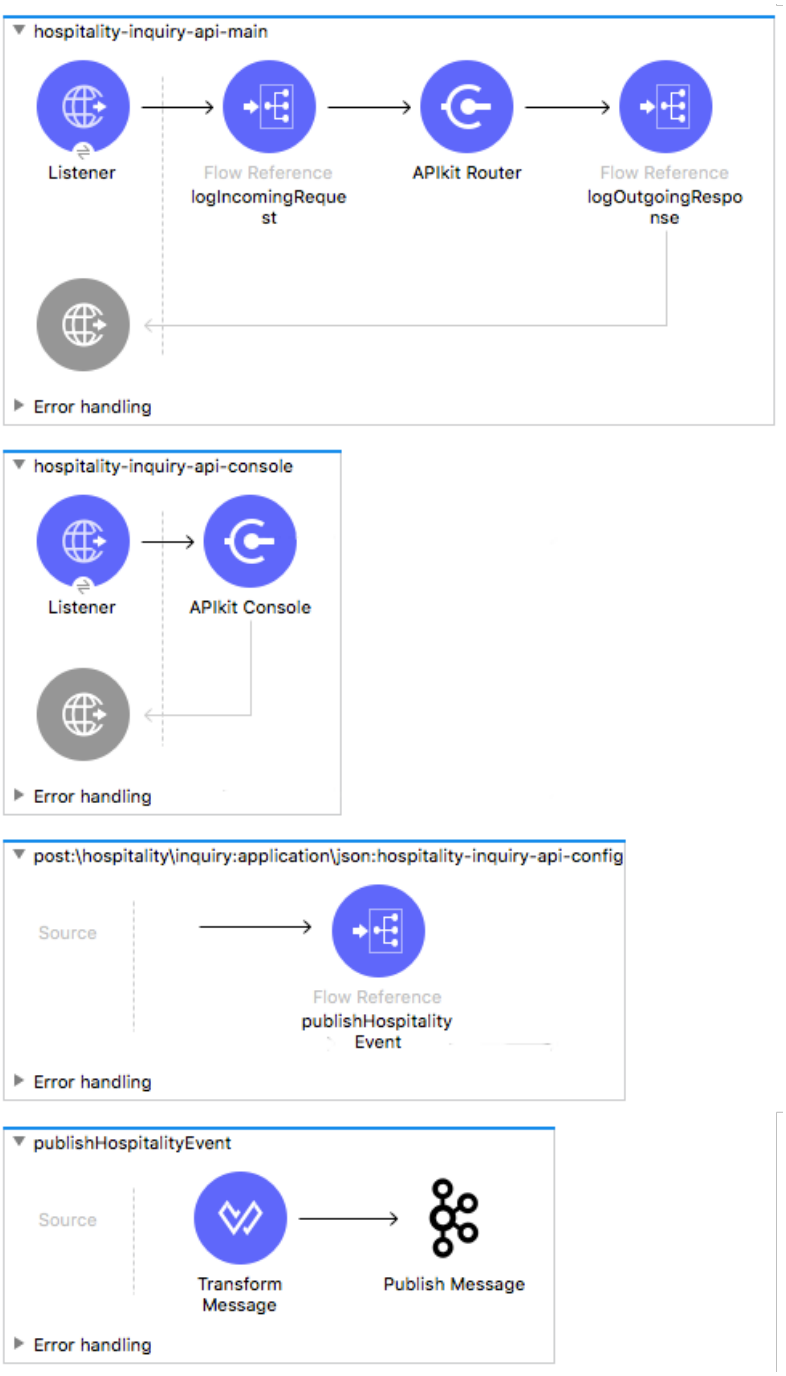

Figura 14: Implementação da API destinada a receber os pedidos de ofertas.

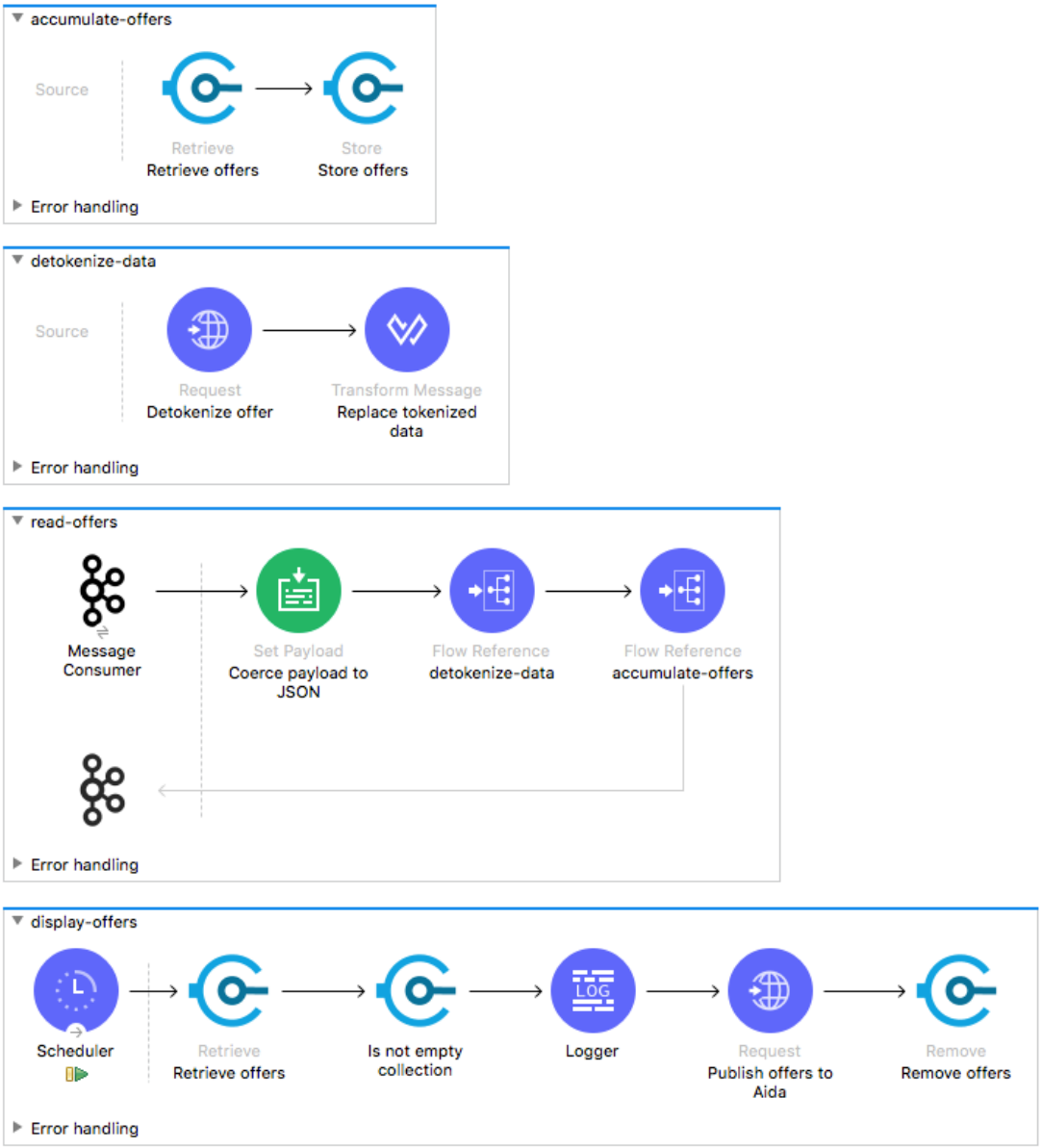

Figura 15: Implementação do acumulador e publicador de ofertas para a aplicação assistente.

5 Disciplinas fundamentais

Para a execução de tarefas anteriormente descritas, foram tomadas como base de conhecimento, principalmente, as disciplinas contidas na Tabela 1

Processamento da Informação	• Elaboração de algoritmos para solucionar problemas • Lógica de programação
Programação Orientada a Objetos	• Python e Java são linguagens orientadas à objetos e a maior parte das atividades do primeiro estágio consistiram em desenvolver aplicações em Python, enquanto Java é necessário para a costumização de comportamentos pré-estabelecidos do servidor de aplicações Mule.
Processamento de Linguagem Natural	• A leitura de arquivos de dados utilizando a biblioteca Pandas foi fundamental para efetuar transformações sobre os dados tratados em cada aplicação.
Análise de algoritmos	• Noções de performance esperada do código escrito e como melhorá-lo
Inteligência Artificial	• Noções de redes neurais serviram para que a proposta de projeto do curso de metodologia ágil fosse elaborada
Banco de Dados	• Criação de tabelas, chaves, índices • Conceito de normalização • Linguagem SQL • Conhecimento geral sobre bancos de dados relacionais
Banco de Dados de Apoio à Tomada de Decisão	• Modelagem da arquitetura de armazenamento de dados • Aplicações de ETL • *Data Warehousing* • Noções de criação de relatórios com ferramentas de visualização de dados • Conceitos de *Business Intelligence* • Estratégias de temporalização de dados
Compiladores	• Conhecer compiladores influenciou a habilidade e a agilidade em depurar programas e mitigar seus defeitos
Linguagens Formais e Autômato	• A experiência com expressões regulares e contribuiu para a criação do filtro de arquivos na aplicação datalake_to_sql_server
Programação para Web	• Conhecimento sobre o protocolo HTTP e sobre RESTful APIs

Sistemas Distribuídos	• Noções de orquestramento de processos e integração de sistemas diversos • Conhecimento fundamental necessário para compreender o papel de um ESB em uma arquitetura de sistemas corporativos
Paradigmas de Programação	• Utilização dos elementos funcionais de Python e Java para escrever código mais limpo e otimizado
Engenharia de Software	• Metodologias de gerenciamento de projetos, em especial as ágeis
Sistemas Operacionais	• Utilização do Linux via terminal
Redes de Computadores	• Endereçamento de computadores em rede e ferramentas de acesso remoto (especialmente SSH) para acessar o *cluster* Hadoop, além de servidores de aplicações na nuvem.

Tabela 1: Matérias relacionadas aos exercícios dos cargos.

6 Considerações finais

Essa seção tem como objetivo mostrar as contribuições dos cargos para as respectivas empresas, para o colaborador e as maiores dificuldades encontradas durante os períodos retratados.

6.1 Contribuições para com a Keyrus

As atividades realizadas durante o intervalo de tempo compreendido entre o início e o fim do estágio supervisionado I beneficiaram a empresa cliente pois, desde o fim da primeira *sprint*, pessoas em cargos gerenciais e de liderança passaram a poder contar o CIP para tomarem decisões mais inteligentes e assertivas sobre os rumos que a empresa trilhará, e a riqueza de dados e de funcionalidades do sistema tende apenas a crescer com o decorrer do tempo.

A experiência também permitiu ao colaborador exercitar uma grande parte do conhecimento adquirido em sua fase de capacitação dentro da universidade, em especial no tangente as áreas de *Business Intelligence*, Engenharia de Dados e escrita de código limpo e manutenível. Além disso, serviu para agregar uma quantia significativa de valor ao seu portfólio de projetos, dado que o CIP é um sistema complexo que utiliza diversas ferramentas de ponta em seu conjunto de tecnologias.

Devido ao tamanho do impacto positivo que o CIP representou para a companhia, e a agilidade com que suas funcionalidades são desenvolvidas e entregues, o time de colaboradores responsáveis por ele recebeu um certificado de reconhecimento do setor global de *Delivery* da AB InBev.

Por fim, trabalhar na construção do CIP permitiu ao estudante aprimorar suas habilidades de trabalho em equipe e a sua compreensão sobre a dinâmica de projetos, de forma que não apenas as suas prospecções de carreira fossem significativamente melhoradas, mas especialmente que a sua carga de experiência pessoal fosse acrescida e, por consequência, sua desenvoltura, tanto na área de tecnologia da informação quanto na esfera pessoal de sua vida.

6.2 Contribuições para com a Avenue Code

A Avenue Code se beneficiou das atividades desenvolvidas pelo colaborador pelos motivos a seguir.

Em primeiro lugar, a consultoria à Samsung deixou-a com uma impressão altamente positiva sobre a Avenue Code, pois foi devido a esta que a campanha de promoções UpHouse foi resgatada de uma falha iminente que causaria prejuízos incalculáveis.

Em segundo lugar, o tempo de estudo individual sobre desenvolvimento Mule, em conjunto com o curso de Mule 4 fornecido pela MuleSoft por meio de sua plataforma online de treinamentos, agregaram mais um membro para o pequeno grupo de profissionais aptos a trabalhar com Mule 4 na Avenue Code. Considerando que uma das prioridades de curto a médio prazo da empresa é a entrada e consolidação de sua posição no mercado internacional de prestação de serviços relacionados

a quarta versão do conjunto de tecnologias fornecido pela MuleSoft, e a dificuldade de encontrar profissionais aptos para tal, a empresa foi capaz de observar de antemão o valor obtido em treinar seus membros para ajuda-la nesta tarefa.

Ademais, o curso *MuleSoft.U Mule 4 for Mule 3 Users* aborda todo o conteúdo necessário para que o aluno esteja apto a ser aprovado no exame de certificação *MuleSoft Certified Developer - Level 1 (Mule 4)*. Dados os objetivos supracitados da empresa contratante e a presença de clientes cada vez mais exigentes nesse mercado, a presença de profissionais certificados em seu corpo de funcionários é altamente relevante. Além disso, um grupo suficientemente numeroso de profissionais certificados contratados pela empresa faz com que a MuleSoft conceda benefícios especiais a esta, como cursos gratuitos, contas com acesso irrestrito às funcionalidades da plataforma, recomendação para clientes etc.

Os ocorridos durante a vigência do estágio supervisionado III, por sua vez, consistiram na primeira atuação profissional de longo prazo destinada a um cliente que o aluno desempenhou pela Avenue Code. Os desafios foram grandes, mas proporcionarão visibilidade global à Avenue Code, que terá seu nome divulgado no MuleSoft Connect 2019, uma conferência destinada principalmente à exposição das características das ferramentas e de oportunidades de negócios envolvendo Mule e seu ecossistema, incluindo consultoria. Como a conferência é principalmente frequentada por pessoas de cargos técnicos elevados nas hierarquias de diversas empresas, esta é uma oportunidade que a Avenue Code terá de sondar o mercado e avaliar prospecções de clientes futuros.

Além disso, o sucesso do projeto culminou no estreitamento da parceria entre a Avenue Code e a MuleSoft, sendo a segunda uma importante aliada para a primeira, considerando os objetivos de médio prazo que esta possui no mercado brasileiro.

Por fim, a experiência vivenciada durante os estágios supervisionados II e III proporcionou ao aluno um conhecimento profundo sobre arquitetura de projetos de software, além do aprimoramento significativo de seus conhecimentos em uma ferramenta com pouca mão-de-obra disponível no mercado, mas com demanda crescente, colocando-o em uma posição privilegiada sob a ótica de sua carreira.

6.3 Dificuldades enfrentadas na Keyrus

A maior dificuldade foi descobrir onde aplicar os conhecimentos de programação e padrões de projeto vistos em sala de aula para resolver os problemas de *Business Intelligence* do mundo real.

As massas de dados possuídas por empresas podem chegar tamanhos imensuravelmente maiores do que as trabalhadas em sala de aula, e tecnologias mais específicas que as abordadas pela universidades são frequentemente necessárias, de forma que é preciso aprender a trabalhar com elas por conta própria. Ademais, não basta que os problemas sejam solucionados. Eles precisam de soluções que utilizem apenas os recursos disponíveis e de forma que seja possível que outros programadores colaborem com - ou continuem - o trabalho, portanto, documentações padronizadas precisam ser escritas e padrões de projetos, seguidos.

6.4 Dificuldades enfrentadas na Avenue Code

Como na Keyrus, um grande desafio foi avaliar de forma correta onde aplicar o conhecimento obtido em sala de aula para resolver os problemas do mundo real, com a diferença de que o produto final a ser entregue é um *Enterprise Service Bus* e não um relatório de BI.

O número de sistemas a serem conectados de forma a trabalharem em conjunto é muito maior do que os casos de integração vistos em Programação para Web e Sistemas Distribuídos, por exemplo. Estes sistemas também tendem a exercer papéis muito diversificados e é atribuição do desenvolvedor de integrações descobrir como trabalhar com todos em conjunto em prol de tornar os requisitos dos clientes em realidade. Além disso, os sistemas costumam expor interfaces de acesso dos mais variados tipos (SOAP, REST, RFC etc). Cada interface troca dados utilizando um formato (JSON, XML, *URL encoded forms*, LDAP, entre outros) e vale-se de um protocolo de comunicação particular, portanto é preciso aprender sobre todas as opções para exercer o papel com sucesso.

O conjunto de ferramentas e padrões arquiteturais que o desenvolvedor de integrações precisa conhecer também é extenso e apresenta uma miríade de aplicações. Alguns exemplos são: filas de mensagens, linguagens de programação, ferramentas de ETL, motores de *containers*, ferramentas de versionamento de código, *frameworks* para o desenvolvimento de aplicações de servidor, paradigma cliente-servidor, conceito de *middleware*, bancos de dados diversos, sistemas operacionais, provedores de computação em nuvem etc. Este extenso ferramental não é coberto em profundidade em sala de aula, de forma que um dos grandes desafios enfrentados foi o de absorver o papel de cada uma delas e como, quando integradas, resolvem determinados (grupos de) problemas.

No geral, integração de sistemas é um tópico pouco abordado na universidade. Sendo assim, a carga de estudo individual requerida para atuar na área é alta.

7 Referências Bibliográficas

[1] Crimson Hexagon. Acessado em 07/12/2018, . URL `https://www.crimsonhexagon.com`.

[2] Documentação da API do Crimson Hexagon. Acessado em 07/12/2018, . URL `https://apidocs.crimsonhexagon.com`.

[3] Documentação da linguagem DataWeave. Acessado em 18/03/2019. URL `https://docs.mulesoft.com/mule-runtime/4.1/dataweave`.

[4] Professor José Finnochio Junior. Acessado em 08/12/2018. URL `http://pmcanvas.com.br`.

[5] MuleSoft.U Mule 4 for Mule 3 Users. Acessado em 20/03/2019. URL `https://training.mulesoft.com/course/mulesoftu-mule443`.

[6] RAML. Acessado em 20/03/2019. URL `https://raml.org`.

[7] Rivery. Acessado em 06/12/2018. URL `https://rivery.io`.

[8] Spring framework. Acessado em 20/03/2019. URL https://spring.io.

[9] Microsoft SQL Server. Acessado em 07/12/2018. URL https://www.microsoft.com/en-us/sql-server/sql-server-2016.

[10] SuperDroid Robots IG42 SB - 4WD - 10 polegadas. Acessado em 28/04/2019. URL https://www.superdroidrobots.com/shop/item.aspx/new-prebuilt-4wd-all-terrain-robot-platform-ig42-sb-with-10-inch-tires/2640/.

[11] W.E. Board. *API Recipes with Mulesoft(r) Anypoint Platform*. White Falcon Publishing, 2017. ISBN 9789386210906. URL https://books.google.com.br/books?id=6CMotAEACAAJ.

[12] David Chappell. *Enterprise Service Bus*. O'Reilly Media, Inc., 2004. ISBN 0596006756.

[13] P.J. Deitel and H. Deitel. *Java How To Program (Early Objects)*. Pearson Education, 2014. ISBN 9780133807943. URL https://books.google.com.br/books?id=34D2AgAAQBAJ.

[14] Y.N. Harari. *Sapiens: A Brief History of Humankind*. Harper Collins, 2015. ISBN 9780062316103.

[15] Zakir Laliwala, Abdul Samad, Azaz Desai, and Uchit Vyas. *Mule ESB Cookbook*. Packt Publishing, 2013. ISBN 1782164405, 9781782164401.

[16] Wes Mckinney. pandas: a foundational python library for data analysis and statistics. *Python High Performance Science Computer*, 01 2011.

[17] G C Rorimpandey, F I Sangkop, V P Rantung, J P Zwart, O E S Liando, and A Mewengkang. Data model performance in data warehousing. *IOP Conference Series: Materials Science and Engineering*, 306(1):012044, 2018.

[18] J. Sutherland and JJ Sutherland. *Scrum: The Art of Doing Twice the Work in Half the Time*. Crown Publishing Group, 2014. ISBN 9780385346467. URL https://books.google.com.br/books?id=93tIAwAAQBAJ.

[19] Guido van Rossum. *Python tutorial, Technical Report CS-R9526*. Centrum voor Wiskunde en Informatica (CWI), 1995.

[20] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2012. ISBN 1449311520, 9781449311520.

Apêndice

Os códigos-fonte não podem ser compartilhados em sua totalidade pois são de propriedade dos respectivos clientes e estão protegidos por lei, no entanto, alguns trechos se encontram a seguir.

A Arquivo main.py do código-fonte do módulo datalake_to_sql_server

```
import sys
import pyodbc
import logging
import properties as p
from ordered_set import OrderedSet
from utils import mkdir_if_not_exists, match_regex, mkdir_hdfs, mv_hdfs, rm_hdfs,
    get_hdfs_file_list


if __name__ == "__main__":

    # making sure that the log directory exists
    mkdir_if_not_exists(p.log_directory)


    # setting up basic log system
    logging.basicConfig(filename=p.log_file,
                        filemode="w",
                        level=logging.DEBUG,
                        format="%(asctime)s - %(levelname)s - %(message)s",
                        datefmt="%Y-%m-%d-%H-%M-%S")
    root_logger = logging.getLogger()


    # adding new handler to write log messages to the standard output as well
    new_log_handler = logging.StreamHandler(sys.stdout)
    new_log_handler.setLevel(logging.DEBUG)
    formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')
    new_log_handler.setFormatter(formatter)
    root_logger.addHandler(new_log_handler)


    root_logger.info("Starting {}".format(sys.argv[0]))


    # listing files under the landing zone
    root_logger.info("Listing files under {}".format(p.landing_zone))
    file_list = get_hdfs_file_list(p.landing_zone, p.file_type_regex[p.data_type])


    # checking which directories are lengthier than allowed
    updatable_dirs = OrderedSet()
```

```
    for f in file_list:
        file_tree = f.split("/")
        parent_dir_name = file_tree[-2]
        if (len(parent_dir_name) > p.dir_max_size):
            # parent directory
            updatable_dirs.add("/".join(file_tree[:-1]))


    # capping their names in 15 characteres
    for d in updatable_dirs:
        dir_tree = d.split("/")
        dir_name = dir_tree[-1]
        new_dir = "/".join(dir_tree[:-1] + [dir_name[:p.dir_max_size]])
        mv_hdfs(d, new_dir)


    # updating reference to the directories post name update
    file_list = get_hdfs_file_list(p.landing_zone, p.file_type_regex[p.data_type])


    # connecting to the database
    root_logger.info("Connecting to the database")
    conn = pyodbc.connect(driver=p.sql_driver_version,
                          server="{},{}".format(p.sql_server, p.sql_port),
                          database=p.sql_database,
                          uid=p.sql_username,
                          pwd=p.sql_password)
    cursor = conn.cursor()


    # making sure that the history zone and the
    # reject directories exist
    mkdir_hdfs(p.history_zone, parent=True)
    mkdir_hdfs(p.reject_dir, parent=True)


    # set of directories under the landing zone
    # that'll be removed when the job concludes
    removable_dirs = OrderedSet()


    # for each file in the JSON file list...
    for lz_file in file_list:

        # compute file metadata
        lz_dir_tree = lz_file.split("/")
        lz_filename = lz_dir_tree[-1]
        hz_file = "/".join([p.history_zone, lz_filename])


        # compute query metadata
        proc_arg = match_regex(p.sql_proc_arg_regex[p.data_type], lz_file)[0]
```

```
94          sql = '{CALL ' + p.sql_procedure + ' (''?'')}'
95
96
97          try:
98              # try running the stored procedure with the
99              # relative (to the project's dir) file path as argument
100             root_logger.info("Running procedure {} with \'{}\' as argument"
101                              .format(p.sql_procedure, proc_arg))
102             cursor.execute(sql, (proc_arg))
103             cursor.commit()
104
105
106             # if the procedure execution is successful, move
107             # the file from the landing to the history zone
108             root_logger.info("Moving {} to {}".format(lz_filename, p.history_zone)
    )
109             mv_hdfs(lz_file, hz_file)
110
111
112             # then add the file's parent and grandparent
113             # directories to the list that'll be removed when empty
114             grandparent_dir = "/".join(lz_dir_tree[:-2])
115             # parent_dir = "/".join([grandparent_dir, lz_dir_tree[-2]])
116             # removable_dirs.add(parent_dir)
117             removable_dirs.add(grandparent_dir)
118
119
120         except pyodbc.ProgrammingError as error:
121             msg_list = [
122                 "Could not run {}".format(p.sql_procedure),
123                 "The argument provided was \'{}\'".format(proc_arg),
124                 "Exit code was \'{}\'".format(error.args[0]),
125                 "Check if the procedure schema and name are correct, " \
126                 "and if all required arguments are present in the call",
127                 "The system's error message was: {}".format(error)
128             ]
129             for msg in msg_list:
130                 root_logger.error(msg)
131
132             if "reject" not in lz_file:
133                 reject_file = "/".join([p.reject_dir, lz_filename])
134                 mv_hdfs(lz_file, reject_file)
135
136
137         except pyodbc.Error as error:
138             msg_list = [
139                 "Could not run {} because of an unidentified error".format(p.
    sql_procedure),
140                 "The argument provided was \'{}\'".format(proc_arg),
141                 "Exit code was \'{}\'".format(error.args[0]),
142                 "Troubleshooting will be required",
143                 "The system's error message was: {}".format(error)
```

```
            ]
            for msg in msg_list:
                root_logger.error(msg)

            if "reject" not in lz_file:
                reject_file = "/".join([p.reject_dir, lz_filename])
                mv_hdfs(lz_file, reject_file)


    conn.close()

    # removing all previously-fulfilled-but-now-empty
    # directories under the landing zone
    for dl_dir in removable_dirs:
        dir_name = dl_dir.split("/")[-1]

        # except if, by any chance, they are a permanent
        # part of the landing zone
        if p.social_media != dir_name and p.media_type != dir_name:
            rm_hdfs(dl_dir, recursive=True)

            logging.info("Removed empty directory from the landing zone: {}"
                         .format(dl_dir))


    logging.info("Finishing {}".format(sys.argv[0]))
    sys.exit(0)
```

B Arquivo main.py do código-fonte do módulo crimson_to_datalake

```
import sys
import logging
import requests
import datetime
import json
import properties as p
from utils import mkdir_if_not_exists, moveFromLocal_hdfs, mkdir_hdfs,
    get_monitors, request_data, build_select_query


if __name__ == "__main__":

    # making sure that the log and data directory exists
    mkdir_if_not_exists(p.log_directory)
    mkdir_if_not_exists(p.data_directory)


    # setting default exit code (if not error occur, this should be used)
    exit_code = 0


```

```
    # making sure that the landing zone exists
    mkdir_hdfs(p.landing_zone, parent=True)


    # setting up basic log system
    logging.basicConfig(filename=p.log_file,
                        filemode="w",
                        level=logging.DEBUG,
                        format="%(asctime)s - %(levelname)s - %(message)s",
                        datefmt="%Y-%m-%d-%H-%M-%S")
    root_logger = logging.getLogger()


    # adding new handler to write log messages to the standard output as well
    new_log_handler = logging.StreamHandler(sys.stdout)
    new_log_handler.setLevel(logging.DEBUG)
    formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')
    new_log_handler.setFormatter(formatter)
    root_logger.addHandler(new_log_handler)


    root_logger.info("Starting {}".format(sys.argv[0]))


    # trying to authenticate
    root_logger.info("Authenticating with username {}".format(p.crimson_username))
    auth_response = requests.get(p.crimson_auth_endpoint)

    auth_json = auth_response.json()
    token = auth_json['auth']
    status = auth_json['status']


    # checking if I am authorized...
    if auth_response.status_code == 200 and token != "" and status == "success":

        root_logger.info("Access granted!")


        # getting the list of monitors
        root_logger.info("Getting the list of monitors from the database")
        monitors_query = build_select_query(p.sql_monitors_table, p.sql_columns, p
.sql_conditions[p.resource])
        monitors, monitors_error = get_monitors(p.sql_server_info, monitors_query)


        # checking if there are monitors registered to query data from
        if len(monitors) != 0 and monitors_error == None:

            root_logger.info("Monitors were successfully retrieved form the
database! We're good to go...")

```

```
71
72          # setting start and end date
73          start_date = (p.exec_time - datetime.timedelta(p.crimson_time_delta)).
    strftime("%Y-%m-%d")
74          end_date = p.exec_time.strftime("%Y-%m-%d")
75
76
77          # selecting which endpoint I should send requests to
78          endpoint = p.crimson_endpoints[p.resource]
79
80
81          # constructing the output with data from all monitors
82          monitors_json = []
83          for monitor in monitors:
84
85              # requesting json data
86              monitor_endpoint = endpoint.format(token, monitor.SocialMediaID,
    start_date, end_date)
87              monitor_response = request_data(monitor_endpoint)
88
89              if monitor_response != -1:
90
91                  # adding identification attributes (these will come from a
    database in the future)
92                  monitor_response['monitor_id'] = monitor.SocialMediaID
93                  monitor_response['brand'] = monitor.BrandName
94                  monitor_response['country'] = monitor.CountryCode
95
96                  monitors_json = monitors_json + [ monitor_response ]
97
98              else:
99
100                 root_logger.warn("Could not retrieve data from monitor {} ({})
     with start date " \
101                                  "{} and end date {}".format(monitor.
    SocialMediaID,
102                                                             monitor.BrandName
    ,
103                                                             start_date,
104                                                             end_date))
105
106
107         # writing output file to the local file system
108         local_file = '/'.join([p.data_directory,
109                                '{}-exec_{}-start_{}-end_{}.json'.format(p.
    exec_time.strftime('%Y-%m-%d-%H-%M-%S'),
110
    start_date,
111
    end_date,
112                                                                 p.
    resource)])
```

```
113
114             with open(local_file, 'w') as outfile:
115                 json.dump(monitors_json, outfile)
116
117
118             # moving output file to the landing zone
119             root_logger.info("Moving {} to the landing zone: {}".format(local_file
    , p.landing_zone))
120             moveFromLocal_hdfs(local_file, p.landing_zone)
121
122
123             root_logger.info("Finishing {}".format(sys.argv[0]))
124
125
126         else:
127
128             root_logger.error("An error occurred while trying to retrieve the " \
129                               "list of monitors to query data from")
130             root_logger.error('Check if there is at least one monitor registered
    in the database')
131             root_logger.error("The system's error message was: {}".format(
    monitors_error))
132             exit_code = 1
133
134
135     else:
136
137         root_logger.error("Crimson API did not authorize the user {}.".format(p.
    crimson_username))
138         root_logger.error("Status code was {}.".format(auth_response.status_code))
139         exit_code = auth_response.status_code / 100
140
141
142     sys.exit(exit_code)
```

C Especificação da API destinada a receber as leituras dos sensores do carro inteligente

```
1  #%RAML 1.0
2  version: v1
3  title: Connected Car Receiver
4
5  mediaType: application/json
6
7  uses:
8    assets: connected-car-assets.raml
9
10 /pressureReading:
11   type:
12     assets.push:
13       datatype: assets.tirePressureReading
```

```
      dataexample:
        carId: gfw37
        timestamp: 2019-04-12T16:55:57.317Z
        tireLocation: FrontLeft
        val: 29.5
        units: PSI
      responsetype: assets.responseMessage
      errortype: assets.errorMessage

/rotationFrequencyReading:
  type:
    assets.push:
      datatype: assets.tireRotationFrequencyReading
      dataexample:
        carId: 863ge
        timestamp: 2019-04-12T16:55:23.157Z
        tireLocation: RearRight
        val: 7.3
      responsetype: assets.responseMessage
      errortype: assets.errorMessage
```

D Especificação da API destinada a receber pedidos de ofertas no início da demonstração de negociação automática

```
#%RAML 1.0
version: v1
title: Hospitality Inquiry API

mediaType: application/json

uses:
  HospitalityTypes: /exchange_modules/hospitalitytypeslib/1.0.1/
    HospitalityTypesLib.raml

types:
  HospitalityInquiry: HospitalityTypes.HospitalityInquiry

  OpenInquiryFormat: HospitalityTypes.OpenInquiryFormat
  HospitalityAllianceCore: HospitalityTypes.HospitalityAllianceCore
  LocalHospitality: HospitalityTypes.LocalHospitality

  ErrorMessage: !include /exchange_modules/errormessage/1.0.2/ErrorMessage.raml

/hospitality/inquiry:
  post:
    body:
      type: HospitalityInquiry
    responses:
      202:
        body:
          type: OpenInquiryFormat | HospitalityAllianceCore | LocalHospitality
```

```
27      400:
28        body:
29          type: ErrorMessage
30      500:
31        body:
32          type: ErrorMessage
```