ARLEN MATEUS MENDES SILVA PINTO

RELATÓRIO TÉCNICO:

PROJETO ARQUITETURAL E DESENVOLVIMENTO DE UMA APLICAÇÃO PARA INTEGRAÇÃO DA PLATAFORMA EQUALS COM ADQUIRENTES

LAVRAS – MG

2021

ARLEN MATEUS MENDES SILVA PINTO

RELATÓRIO TÉCNICO:
PROJETO ARQUITETURAL E DESENVOLVIMENTO DE UMA APLICAÇÃO PARA INTEGRAÇÃO DA PLATAFORMA EQUALS COM ADQUIRENTES

Relatório técnico apresentado à Universidade Federal de Lavras, como parte das exigências do curso de Sistemas de Informação, para obtenção do título de Bacharel

Prof. DSc. Maurício Ronny de Almeida Souza
Orientador

LAVRAS – MG

2021

ARLEN MATEUS MENDES SILVA PINTO

RELATÓRIO TÉCNICO: PROJETO ARQUITETURAL E DESENVOLVIMENTO DE UMA APLICAÇÃO PARA INTEGRAÇÃO DA PLATAFORMA EQUALS COM ADQUIRENTES

Relatório técnico apresentado à Universidade Federal de Lavras, como parte das exigências do curso de Sistemas de Informação, para obtenção do título de Bacharel

APROVADA em 30 de abril de 2021.

Prof. DSc. Maurício Ronny de Almeida Souza	UFLA
Prof. DSc. Renata Teles Moreira	UFLA
Guilherme Camilo Rezende	Equals

Prof. DSc. Maurício Ronny de Almeida Souza
Orientador

LAVRAS – MG
2021

Dedico, tanto este trabalho quanto tudo que estou me tornando como profissional, à minha mãe. E também à minha avó Madalena (in memoriam), cujos ensinamentos serão eternos.

AGRADECIMENTOS

Agradeço à Deus, por ter me capacitado de estar aqui e realizar a graduação.

Agradeço à minha mãe, Rita de Cassia, por tudo que fez por mim e por ser o pilar mais forte que eu tenho na vida, por todos os sacrifícios, os momentos de mãe e também de amiga, o apoio com fé e amor a todas as minhas decisões e por ter me trazido até aqui. Minha inspiração.

Aos meus irmãos, Renato, Eduardo e Lays, por serem, sempre que possível, presentes, pela família unida que somos e pelos sobrinhos maravilhosos.

À minha namorada Carolina pelo apoio e companhia durante a conclusão deste momento único.

Aos amigos que Lavras e a UFLA me deram. Em especial ao Thiago, que esteve comigo desde o primeiro dia que cheguei aqui.

À UFLA, em especial ao Departamento de Ciências da Computação, pela estrutura e profissionais que me instruíram e me capacitaram.

À Renata Teles, por ser mais que uma professora, ser uma amiga e conselheira e sempre ter acreditado em mim e me apoiado.

À Comp Junior, por ter me ensinado a ir além e me impulsionado tanto a crescer. Pelas amizades feitas ali e por todos os momentos incríveis.

Por fim, ao meu professor e orientador Maurício Ronny pelas aulas, pelos ensinamentos durante a graduação e por me auxiliar e guiar na escrita deste trabalho.

MUITO OBRIGADO!

RESUMO

Este relatório técnico tem como objetivo descrever a concepção, coleta de requisitos, desenvolvimento e implementação da arquitetura de um sistema, tendo como principal função integrar o Equals (gestão financeira inteligente) com os sistemas de meios de pagamentos. O sistema, denominado *E-downloader*, foi projetado e arquitetado em uma arquitetura de microsserviços, utilizando da linguagem Javascript para implementação dos serviços e tendo dois padrões arquiteturais de estrutura da aplicação mais dominantes, que serão descritos ao longo da explanação da solução. Cada microsserviço tem a finalidade de se comunicar com uma adquirente, podendo até multiplicar um microserviço para realizar a integração de forma paralela, dependendo do volume de dados. A implementação deste sistema traz maior segurança, controle e confiabilidade para a Equals (empresa que leva o mesmo nome da plataforma e responsável pela mesma) em relação a estas integrações.

Palavras-chave: Arquitetura de Software. Microsserviços. Aplicações Web. Integração de Aplicações

ABSTRACT

This technical report aims to describe the design, requirements collection, development and implementation of a system architecture, with the main function of integrating Equals (intelligent financial management) with payment methods systems. The system, called E-downloader, was designed and architected in a multi-service architecture, using the Javascript language for the implementation of services and having two most dominant architectural patterns of application structure, which will be described throughout the explanation of the solution. Each microservice has the purpose of communicating with an acquirer, and can even multiply a microservice to perform the integration in parallel, depending on the volume of data. The implementation of this system brings greater security, control and reliability to Equals (a company that bears the same name as the platform and responsible for it) in relation to these integrations.

Keywords: Software Architecture. Microservices. Web Applications. Application Integration

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

A cada minuto, o mercado financeiro torna-se mais digital, e por consequência os meios de pagamentos digitais passam a representar uma parcela considerável das transações financeiras realizadas a partir de vários terminais, seja na compra com cartão a partir de uma maquininha ou mesmo em um e-commerce. Essa crescente modernização engendra uma gama de adquirentes que fazem todo o intermédio dessas transações, conectando os bancos, bandeiras e comércio. Contudo, o lojista pode encontrar certa dificuldade em gerenciar tantas informações, atravancando sua gestão de valores tanto a receber quanto a pagar de taxas, fazendo com que tais valores não fiquem claros para ele. É exatamente nesse espaço que se situa o objeto da empresa Equals.

A Equals (EQUALS, 2021), que atua há 10 anos no mercado, foi fundada com o objetivo de desenvolver soluções para melhorar a gestão financeira das empresas. Com uma plataforma que leva o nome da empresa, é responsável por conciliar e simplificar informações sobre transações para o lojista e centralizá-las, visto que cada lojista pode, inclusive, ter mais de um meio de captura destas transações. Atualmente, com lojistas que atingem milhares de transações diariamente, o Equals - aqui já sendo referido o produto - se tornou imprescindível. Consequentemente, a Equals vem se revelando como grande conciliadora financeira, trazendo cada vez mais detalhes e funcionalidades em seu sistema conciliador.

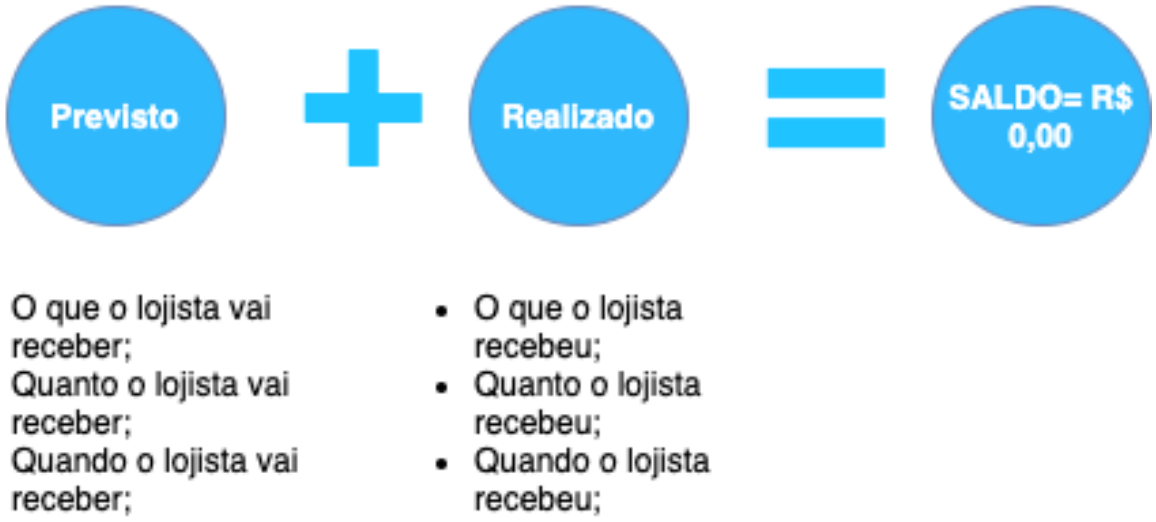
O Sistema Equals tem por objetivo realizar a conciliação das transações e seus valores, demonstrar informações sobre agenda financeira e valores reais e líquidos a serem recebidos pelo lojista e, também, gerar um informativo sobre tudo que está sendo pago de taxas e encargos, criando uma agenda financeira muito bem montada e detalhada. O propósito final é entregar ao lojista uma gestão financeira completa e relevante do seu negócio.

Buscando um melhor desenvolvimento do trabalho, elucida-se que as transações faladas até aqui não são nada mais do que vendas. Uma venda pode ser definida como a transferência da posse de alguma coisa mediante pagamento de certa quantia. Uma venda pode ser caracterizada por uma venda à vista, venda à crédito parcelada pelo lojista, venda à crédito parcelada pela adquirente, venda por boleto, dentre outras hipóteses.

Já o pagamento é a liquidação da venda, ou seja, a confirmação da entrega da quantia devida. Neste momento, podem ser realizadas outros tipos de transação, derivados de uma venda, como cancelamentos, parcelamentos e *chargebacks* (EQUALS-BLOG, 2021). As mencionadas derivações também são englobadas no conciliador Equals.

Na figura 2.1, busca-se demonstrar de forma resumida como ocorre a conciliação. O saldo da diferença entre os valores previstos e os valores efetivamente realizados deve ser sempre zero. Caso ocorra alguma divergência nesse valor, acusa-se erro, que pode ser uma taxa cobrada de forma errônea, alguma venda cancelada, valores a serem estornados, ou outra possibilidade que deverá ser analisada no caso concreto.

Figura 1.1 – Vendas e pagamentos



Fonte: Arlen M. Mendes

1.1 O Problema

A Equals atende uma quantidade enorme de clientes dos mais diversificados mercados. Todos estes realizam vendas a partir de uma ou mais adquirentes[1]. Este trabalho só é possível a partir da coleta direta de informações destas adquirentes. Como elas são independentes, cada uma demonstra cada dado de uma forma, trazendo assim uma dificuldade de interpretação para o cliente final. Então, a Equals recolhe as informações de todos os clientes em todas as adquirentes que integram a ela, padroniza esta informação e demonstra para o usuário da plataforma.

Estas informações, por muito tempo, eram disponibilizadas quase que exclusivamente a partir do envio de arquivos para a conciliadora seguindo um intervalo de tempo padronizado. Em alguns casos era

[1] As "maquininhas"de cartão são as famosas adquirentes no mercado, responsáveis pela comunicação de uma transação. Elas analisam, processam e liquidam operações de cartões, sejam Pix, de débito ou crédito.

necessário realizar a "busca"deste arquivo, como acessar um repositório de arquivos ou servidor de dados. Esta segunda forma era bem esporádica e incomum, fazendo com que medidas paliativas e pouco planejadas fossem tomadas para adquirir estes dados/arquivos, como a criação de pequenas aplicações e/ou *scripts* de linha de comando em sistemas Linux.

Desde o início do Equals, todas as informações necessárias chegam através de arquivos enviados pelas adquirentes. O mencionado conciliador foi projetado para ler estes arquivos, sejam no formato csv, posicional (com um arquivo auxiliar enviado pela adquirente para descrever o que cada posição significa), XML ou JSON. Cada adquirente possui sua rotina de envio das informações, podendo o envio ser diário, semanal ou quinzenal.

A partir do recebimento de tais arquivos, o Equals tem rotinas para, além da leitura e obtenção das informações contidas neles, catalogá-los e guardá-los para futuras auditorias e possíveis consultas.

Antigamente, as adquirentes enviavam através de arquivos as informações necessárias para realizar a conciliação. Contudo, atualmente, com o aumento da demanda e surgimento de novas plataformas, estas integrações passaram a ser a partir de consumo de API[2] (REDHAT, 2021b). Entretanto, o Equals não foi projetado para este cenário, visto que ele praticamente não existia. Não que as API não tenham sido inventadas naquele tempo, mas era um processo que não era praticado por nenhum dos dois lados - nem pela Equals, nem pelas adquirentes.

Portanto, uma mudança no Equals para operar com esta forma de consumo de dados e integração se mostrou um processo inviável, visto que há outros processos além da alimentação da aplicação, que o formato de arquivos faz parte e não poderia ser afetado caso fosse pensada uma solução em que houvesse alteração na aplicação Equals para operar dessa forma. Em capítulo posterior, este trabalho apresentará a possível solução para o presente obstáculo.

1.2 Objetivo

O objetivo deste trabalho é descrever o *E-downloader*, uma aplicação desenvolvida para intermediar os dois cenários descritos como problemas na seção anterior. De um lado temos novas formas de obtenção da informação, que passaram a ser a partir de consultas. E do outro, o Equals que está preparado para receber arquivos previamente definidos e bem organizados para realizar a conciliação e demonstração das informações. Logo, o *E-downloader* é uma aplicação responsável por realizar a consulta dos dados, escreve-

[2] *Application Programming Interface* - um conjunto de rotinas e padrões de programação para acesso a um aplicativo de software ou plataforma

los em arquivos e enviá-los ao Equals para que possa seguir o fluxo normal e já bem consolidado de obtenção das transações.

1.3 Organização do Trabalho

Além deste capítulo introdutório, este trabalho está organizado nos seguintes capítulos. O Capítulo 2 apresenta conceitos e tecnologias utilizados para o desenvolvimento da solução. O Capítulo 3 descreve todos os detalhes da solução e a forma como cada tecnologia é aplicada no seu desenvolvimento. O Capítulo 4 tece as considerações finais.

2 CONCEITOS E TECNOLOGIAS UTILIZADAS

Neste capítulo, busca-se explanar sobre as tecnologias envolvidas no desenvolvimento da solução, nos conceitos propostos e utilizados e ferramentas que irão apoiar as atividades específicas que a solução terá de desempenhar.

2.1 Arquitetura e Ambiente

A aplicação que visa a solução está sendo desenvolvida em uma arquitetura de microsserviços (REDHAT, 2021a). O princípio básico dos microsserviços é a simplicidade. Quando o aplicativo é dividido em um conjunto de fragmentos compostos menores, eles são mais fáceis de criar e manter. Cada função é denominada um serviço e pode ser criada e implementada de maneira independente. Como cada microsserviço é, na verdade, um segmento de código separado, o problema de gerenciamento de código é reduzido. Diferentes linguagens de programação, bancos de dados e ambientes de software podem ser usados para implementar serviços. Desta forma, isso permite a implantação, reconstrução, reimplantação e gerenciamento independentes de cada serviço, também simplifica e facilita a escalabilidade[1] de funções de forma mais independente.

Por consequência, os microsserviços permitem a criação de produtos ao invés de projetos. Na verdade, a arquitetura de microsserviço convida a equipe a se concentrar na criação de funções de negócios.

2.2 Tecnologia para o Desenvolvimento

Todas as API's desta solução e seus microsserviços estão sendo desenvolvidos com a linguagem Javascript na plataforma Nodejs (FOUNDATION, 2021b), que é uma *Engine* baseada na V8 (o motor javascript do Google Chrome), permitindo que trabalhemos com javascript do lado do servidor.

Ainda, toda esta estrutura gira em torno de um banco de dados que contém todas as informações de quais clientes estão vinculados a quais adquirentes. Os SGBDs (Sistemas de Gerenciamento de Banco de Dados) (BATISTELLA, 2021) são conjuntos de softwares utilizados no gerenciamento de uma base de dados. Estes ajudam o administrador do banco de dados a organizar, proteger, editar e acessar as informações da empresa armazenadas no banco de dados. O SGBD (Sistema de Gerenciamento de Banco de Dados) utilizado pela aplicação que visa a solução é o postgres(POSTGRES, 2021).

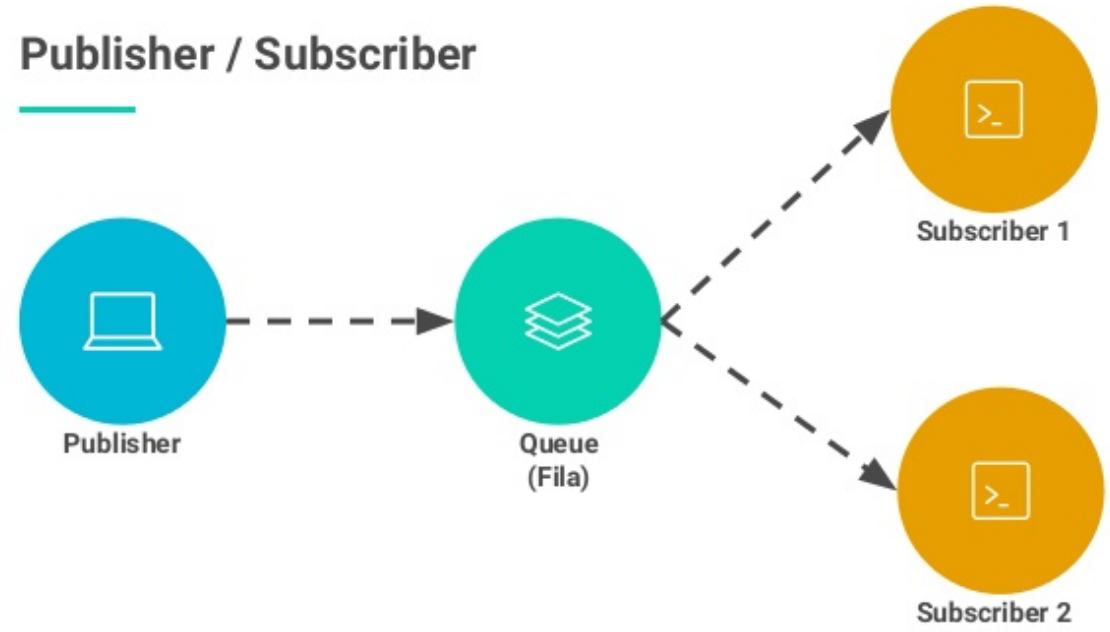
[1] Escalabilidade é a habilidade de ajustar o sistema à capacidade desejada, o que geralmente significa lidar com mais e mais cargas de trabalho com baixo custo.

2.3 Sqs - sistema de mensageria

O Sqs (SEVICES, 2021) é um sistema de mensageria. Mensageria é um conceito que define que sistemas distribuídos possam se comunicar por meio de troca de mensagens (evento), sendo estas mensagens gerenciadas por um *Message Broker* (servidor/módulo de mensagens) que nada mais é que um servidor de mensagens, responsável por garantir que a mensagem seja enfileirada e armazenada em disco (opcional), garantindo que ela fique lá enquanto necessário até que alguém (*Subscribes* ou *Consumer*) a retire de lá.

Neste tipo de serviço há dois personagens, os *Publishers* e os *Subscribes*. Um *Publisher* é responsável por enviar mensagens para a fila do Sqs os Subscribes são responsáveis por observar essas filas e consumir as mensagens quando houver alguma. A figura 3.1 representa o processo.

Figura 2.1 – Filas



Fonte: Arlen M. Mendes

2.4 Redis

O Redis (REDISLAB, 2021) é um sistema de Banco de Dados em memória que nos ajuda com Cache para otimizar o processo e evitar um uso excessivo do SGBD e assim otimizar nosso trabalho. Ainda, é utilizado em algumas soluções dentro de nossos microsserviços como tolerância a falha, em caso de erro

ao buscar os dados de algum cliente ou algo do tipo e, por exemplo, quando ocorra algum problema, para depois realizar tratativas para mitigar este erro a partir de operações que estejam pré-definidas.

Por conta da sua velocidade e facilidade de uso, o Redis é uma escolha que está em alta demanda para aplicações web e móveis, como também de jogos, tecnologias de anúncios e IoT, que exigem o melhor desempenho do mercado.

2.5 Docker

Docker (INC., 2021) é um conjunto de produtos que usa virtualização de nível de sistema operacional para entregar software em pacotes. Ele permite que os usuários criem ambientes independentes e isolados para iniciar e implantar seus aplicativos. Esses ambientes são chamados de *containers*.

Um *container* é um método utilizado na implantação e execução de aplicativos distribuídos sem a necessidade de configuração de uma máquina virtual completa para cada um deles. Isso permitirá que o desenvolvedor execute um *container* em qualquer máquina.

2.6 Kubernetes

Toda a aplicação será executada no Kubernetes (FOUNDATION, 2021a). Kubernetes (ou K8s, como é comumente chamado dentro da comunidade de desenvolvimento) é um produto *Open Source* utilizado para automatizar a implantação, o dimensionamento e o gerenciamento de aplicativos em *containers*. Atualmente, o Kubernetes é mantido pela *Cloud Native Computing Foundation*.

3 A SOLUÇÃO

Neste capítulo, está descrita a solução técnica e tecnológica proposta e que está em desenvolvimento no momento da disponibilização deste trabalho, para suprir todas as necessidades e cenários que surgiram ao longo do tempo e que foram descritas anteriormente.

3.1 Concepção da Ferramenta

Durante a análise dos cenários de integrações com todas as adquirentes que disponibilizam API's para consumo, foram detectados vários pontos que são comuns, principalmente por essas aplicações serem REST (REDHAT, 2021c). Também, cada aplicação tem uma forma de realizar autenticação, demonstrar estes dados e tempo de disponibilização dos mesmos. Assim vimos que seria necessário a criação de uma aplicação que intermedie essa integração,

Assim, foram levantados os requisitos das aplicações.

3.1.1 Requisitos Funcionais

Os requisitos funcionais da aplicação são listados a seguir.

- Deve ser acessado a API da adquirente e obtidos os dados das movimentações financeiras.
- Os dados devem ser escritos em arquivos com nomes padronizados e extensões devidamente identificadas.
- Deve haver um controle de cadastros para clientes que realizem transações com as adquirentes integradas.
- Deve haver uma relação que especifique todas as adquirentes que cada cliente utilize.
- Cada adquirente tem suas particularidades, principalmente de acesso, então, é necessário haver painéis de configuração para cada adquirente caso haja configurações dinâmicas.
- Deve haver um controle da periodicidade com que são buscados os dados nas adquirentes, visto que podem ocorrer alterações nessas datas.
- Deve haver suporte, ou até uma migração, das aplicações que já existem para a arquitetura cetralizada e controlada.

- As aplicações devem se comunicar de forma única com as adquirentes, visto que cada uma tem suas regras e padrões
- Adquirentes que não forneçam os arquivos em si, mas apenas os dados, são necessários tratamentos para que gere um arquivo padronizado para envio ao Equals.

3.1.2 Requsitos Não-Funcionais

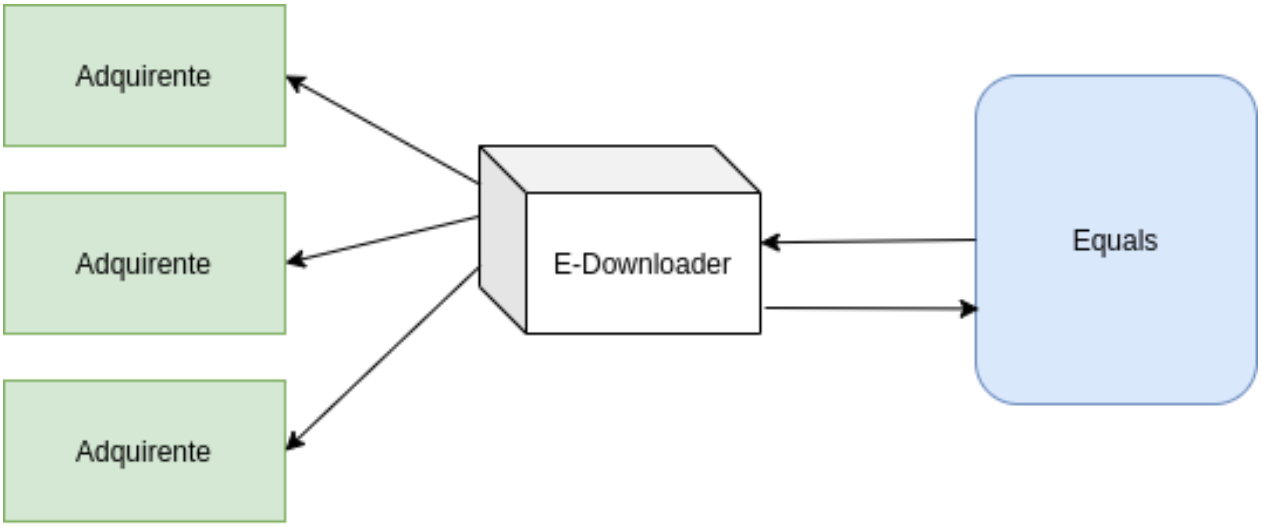
Os requisitos não-funcionais da aplicação são listados a seguir.

- Cada adquirente tem sua própria regra de acesso e envio dos dados, então devem ser tratadas separadamente.
- Deve ser possível realizar uma escala da aplicação de forma fácil, devido a altos volumes de dados.
- Devido ao grande número de adquirentes comtempladas neste cenário, deve haver uma forma de controlar estas integrações de forma mais automática. Com automações de *deploy* e ciclo de vida das aplicações.
- Deve haver tolerância a falhas, principalemnte quando houver problemas com as API's das adquirentes, para poder tentar realizar as operações de busca mais de uma vez, caso necessário.
- Deve haver uma forma de notificar a equipe responsável pelas aplicações caso algo grave ocorra, como perda de conexão com os bancos de dados, em tempo real.

3.2 Arquitetura da Solução

Há um volume enorme de adquirentes que são necessárias às integrações hoje a partir de API's, e cada dia surgem novas e as já existentes vem se atualizando para este modo de integração. Logo, um controle centralizado para gerenciar estas integrações e ao mesmo tempo uma atuação independente na interação com cada uma das adquirentes, nos levou a uma arquitetura de microsserviços. Como cada adquirente tem a sua comunicação e estrutura de dados diferente, precisamos tratar cada uma delas de forma particular. Algumas adquirentes tem um volume de dados maior, clientes com um volume de vendas imenso, outros não, fazendo-nos ter que escalar cada uma dessas aplicações de forma independente para uma melhor otimização do trabalho e menores custos no final. Para ilustrar, na figura 3.1, vimos como a solução deve se comportar, fazendo um intermédio das adquirentes com o Equals e possibilitando a integração.

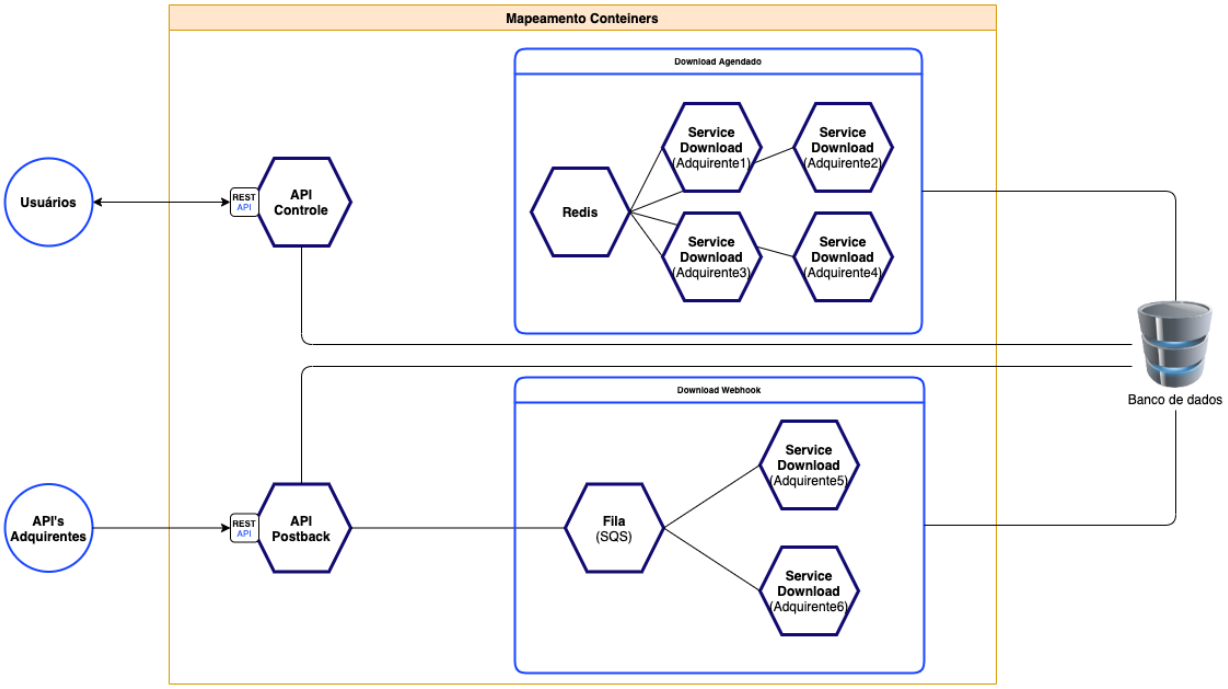
Figura 3.1 – E-downloader



Fonte: Arlen M. Mendes

Na figura 3.2, demonstra-se como se dá toda essa estrutura. Depois, nas seções a seguir, será descrito cada particularidade relevante de todos estes componentes que compõem todo o projeto. A solução proposta aqui é voltada para o backend, e tem pouca especificação para o frontend pois a interação humana é mínima.

Figura 3.2 – Arquitetura



Fonte: Arlen M. Mendes

3.3 Categoria dos microsserviços

Basicamente, temos dois tipos de microsserviços que estão sendo implementados. Proativos e reativos.

Os proativos são aqueles que, em um determinado momento (configurado de forma personalizada para cada adquirente), irá buscar os dados de vendas de um determinado intervalo de tempo na adquirente ao qual ele foi desenvolvido para integrar. Ilustra-se: todo dia às 7h da manhã, o microsserviço "A"vai consultar a API da Adquirente "1"em busca dos dados dos clientes Equals que efetuam vendas com esta adquirente. Na fugura 4.1, são identificados como Download agendado.

Já os reativos, são microsserviços que irão aguardar que a aplicação das adquirentes notifique a Equals, através de uma API disponibilizada pela conciliadora para este fim, que os dados dos clientes que utilizam do Equals estão prontos e podem ser consumidos/buscados. Na fugura 4.1, são identificados como Download *Webhook*.

3.3.1 Microsserviço Agendado

O funcionamento dos microsserviços é integrar com as API's das adquirentes (cada serviço é responsável por uma adquirente diferente), coletar as informações financeiras das vendas dos clientes, escrever em arquivos devidamente identificados e catalogados e enviar para o servidor de processamento do Equals. Como cada cliente estará devidamente cadastrado na base de dados, e todas informações necessárias para esta interação, como credenciais de acesso, frequência com que os dados devem ser buscados e etc., estarão atuando de forma independente. Esta aplicação apresenta 4 camadas, como representada na figura 3.3.

A camada de Cron[1] é responsável por iniciar o acesso a API da adquirente de acordo com a configuração previamente realizada.

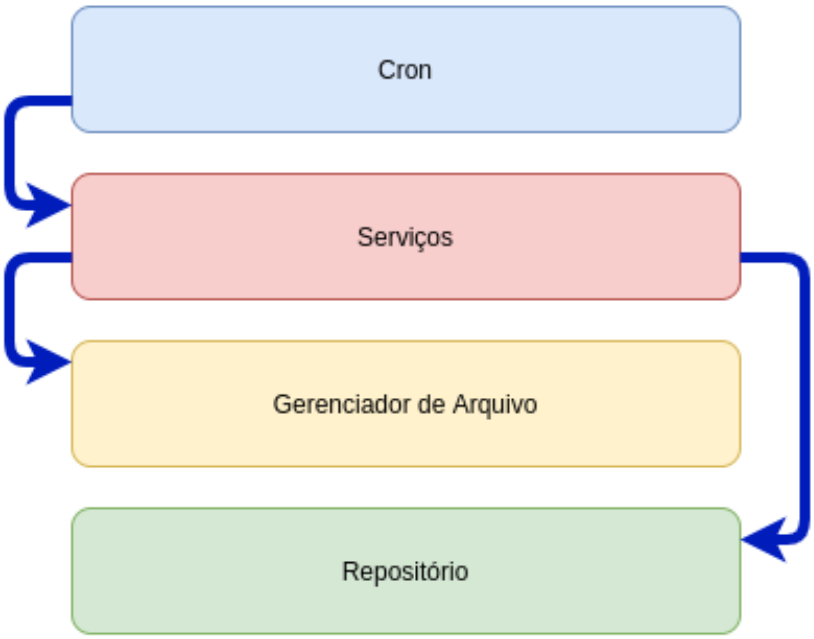
A camada de serviço realiza o acesso a API e aplica as regras de negócio necessárias de acordo com as informações buscadas no banco de dados através dos Repositórios. Após isso, aciona o gerenciador de arquivo para manipular arquivos com os dados buscados.

Já a camada de Gerenciador de Arquivo é responsável por realizar a escrita dos arquivos e também por movê-los para o Equals, para que possam ser manipulados após o fim da escrita.

[1] O utilitário de software cron, também conhecido como cron job, é um agendador de tarefas baseado em tempo em sistemas operacionais de computador semelhantes ao Unix.

Por último, a camada de Repositórios, é uma abstração de acesso a base dados. Ela é responsável por implementar funções que serão disponibilizadas para que a aplicação possa interagir com o SGBD.

Figura 3.3 – ArquiteturaMicrosserviçoAgendado



3.3.2 Microsserviço Reativo

O microsserviço Reativo é desenvolvido em uma Arquitetura Orientada a Eventos[2], que faz com que a aplicação seja acionada através de eventos. Um evento é qualquer ocorrência importante ou mudança de estado no software ou hardware do sistema. Notificação de evento é uma mensagem enviada pelo sistema para notificar outra parte do mesmo sistema que algo aconteceu. Utilizamos do SQS para receber estes eventos e assim realizar as devidas operações de obtenção dos dados. O microsserviço Reativo foi projetado em 4 camadas, representado na figura 3.4.

A camada SQS *Consumer* é responsável por receber os eventos e acionar a camada de serviço para realizar as devidas operações.

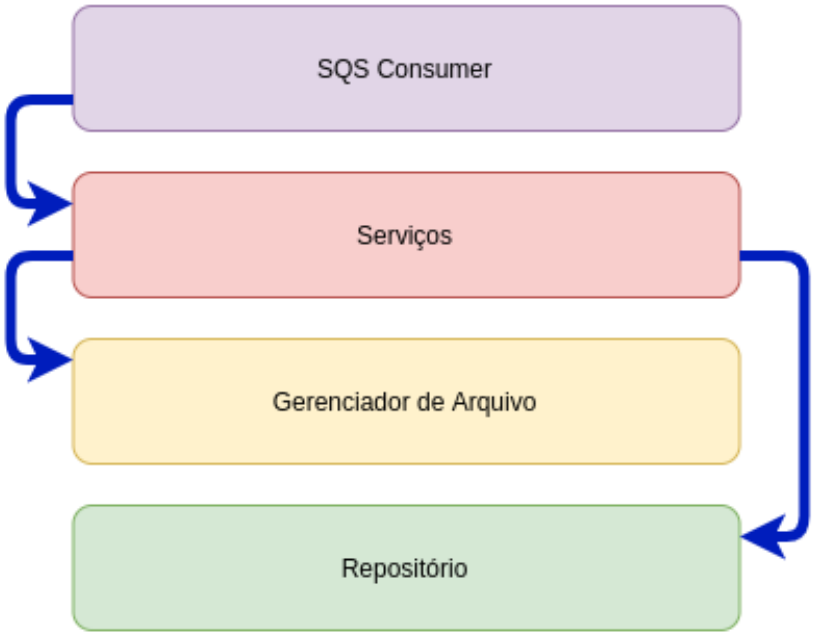
A camada de serviço realiza o acesso a API e aplica as regras de negócio necessárias de acordo com as informações buscadas no banco de dados através dos Repositórios. Após isso, aciona o Gerenciador de arquivo para manipular arquivos com os dados buscados.

[2] https://www.redhat.com/pt-br/topics/integration/what-is-event-driven-architecture

A camada de Gerenciador de Arquivo é responsável por realizar a escrita dos arquivos e também por movê-los para o Equals, para que possam ser manipulados, após o fim da escrita.

A camada de Repositórios, é uma abstração de acesso a base dados. Ela é responsável por implementar funções que serão disponibilizadas para que a aplicação possa interagir com o SGBD.

Figura 3.4 – ArquiteturaMicrosserviçoReativo



3.4 API Controle

Será uma API responsável por gerenciar todos os clientes e a quais adquirentes cada um deles está vinculado. Esta base de dados será consumida por todos os microsserviços, contendo as particularidades, configurações e dados para acessar as API's das adquirentes. Esta aplicação seguirá o padrão REST (REDHAT, 2021c).

A aplicação apresenta 5 camadas sequencialmente ilustradas na figura **??**. Dessa forma cada camada é responsável por uma etapa do ciclo de vida de cada requisição feita a ela.

A camada de Rota (ou roteamento) realiza a verificação através da URI[3] e direciona para o controlador responsável.

[3] Em português: Identificador de Recursos Universal, como diz o próprio nome, é o identificador do recurso. Basicamente indica o que o Usuário da aplicação está tentando realizar na API.

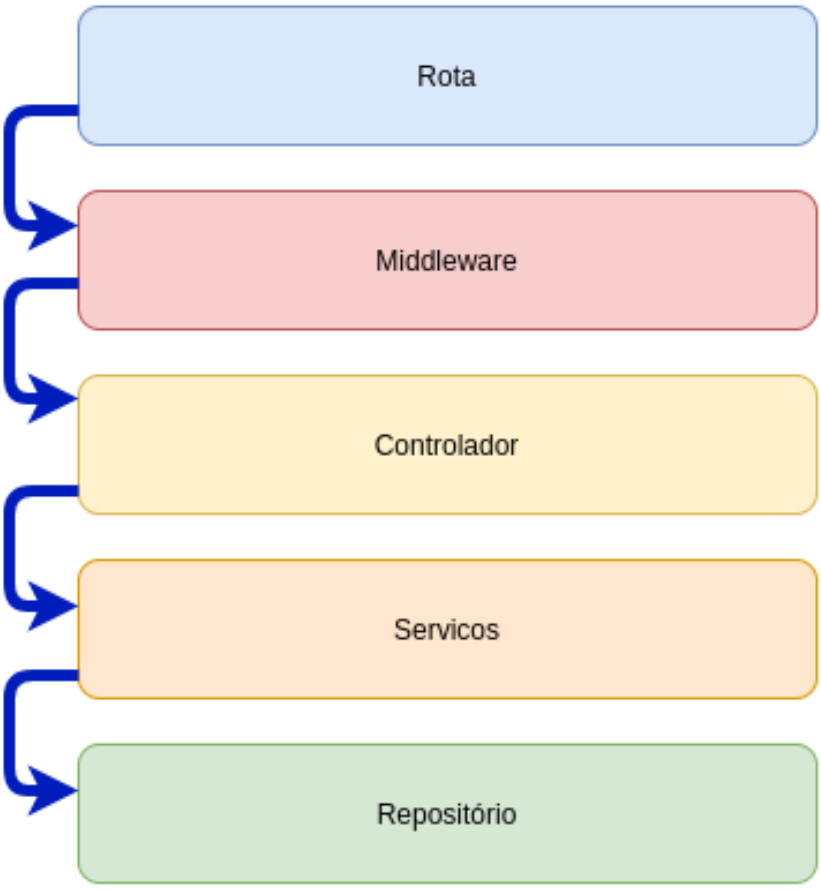
A camada de *middleware* realiza a validação de dados e parâmetros enviados na requisição, evitando assim processamentos errados com dados inválidos e até mesmo ataques como *SQLInjection*.

A camada de Controladores (Controlador) realiza o direcionamento para o Serviço que realizará a ação solicitada pelo usuário. Após, de acordo com o retorno deste serviço, prepara uma resposta sobre o resultado da operação.

A camada de Serviço é responsável por realizar a atividade solicitada pelo usuário e aplicar todas as regras de negócio que ali são necessárias. Em caso de necessidade de buscar, apagar ou adicionar dados à base de dados, realiza-se interações com o Repositório.

A camada de Repositórios é uma abstração de acesso a bases de dados. Ela é responsável por implementar funções que serão disponibilizadas para que a aplicação possa interagir com o SGBD.

Figura 3.5 – ArquiteturaApiControle



3.5 API Postback

A API de *postback* é responsável por receber as notificações emitidas pelas adquirentes informando que os dados de determinados clientes estão prontos e podem ser buscados pela Equals. Assim, a API de *postback* joga as informações no sistema de filas (SQS) para que os microsserviços reativos possam atuar.

As mensagens contidas no Sqs basicamente contém informações previamente padronizadas e que, ao serem consumidas, informam ao microsserviço que os dados de determinado cliente estão disponíveis. Utiliza-se do Sqs pois, como uma de suas ferramentas é a persistência da mensagem até que seja consumida, garante-se que não serão perdidas informações quando uma adquirente emitir uma notificação e algum microsserviço esteja fora do ar. Dessa forma, a informação será consumida imediatamente quando o serviço estiver disponível. A Arquitetura desta aplicação é idêntica à apresentada na figura 3.5, tendo um serviço para enviar as mensagens ao SQS.

3.6 Sistema de Cache = Redis

Utiliza-se do Redis para guardar informações temporárias para realizar operações com mais rapidez devido ao acesso ser mais ágil do que o banco de dados tradicional. Um exemplo do uso desta ferramenta é para realizar uma das atividades de tolerância a falhas, como quando o acesso a API de alguma adquirente falhe para algum cliente. Assim, esta informação é armazenada para uma tentativa posterior, sem a necessidade de buscar no SGBD informações de todos os clientes para verificar se a busca dos dados foi realizado com sucesso ou não.

4 CONCLUSÃO

Este relatório técnico descreveu o sistema *E-Downloader* em relação às tecnologias adotadas, requisitos atendidos, suas principais funcionalidades e a visão geral de sua arquitetura. O desenvolvimento da arquitetura desta solução foi realizado pelo autor deste trabalho no período de novembro de 2020 a fevereiro de 2021.

O sistema em desenvolvimento vem atendendo as expectativas esperadas e está em constante implementação e evolução. Mostrou-se também uma forma bem construída de abordar tal cenário, visto que pouco impactou na plataforma, considerando a forma de se processar os dados. Com esta maneira de abordar as integrações, a rotina de trabalho com o Equals pelo lado dos demais times de desenvolvimento foram minimamente impactados, com alguns times não tendo sido sequer impactados. A implementação desta solução vem trazendo um maior conforto quanto ao controle das integrações com API's, já que mostra um padrão de comportamento dos microsserviços e traz uma vasta base de dados de status das aplicações e até mesmo das plataformas das adquirentes.

Há projetos e expectativas de utilizar algumas tecnologias elencadas nesta solução, como o Kubernetes, em outras partes da plataforma, levando em consideração os ganhos que tivemos e o controle das aplicações em ambientes minuciosamente pensado devido aos benefícios dos containers.

Em suma, a concepção dessa ferramenta utilizou conhecimentos adquiridos durante o curso de Bacharelado em Sistemas de Informação, principalmente nas disciplinas de Engenharia de Software, Computação em Nuvem, Sistemas Distribuídos e Gerência de Projetos. A disciplina Sistemas Distribuídos, devido ao conhecimento sobre aplicações web e servidores de hospedagens, já a disciplina de Gerência de Projetos, devido ao conhecimento sobre planejamento de sistemas e análise de custo x benefício, foram particularmente importantes.

REFERÊNCIAS

BATISTELLA, C. **SGBD**. 2021. Acessed: 2021-04-05. Disponível em: <https://www.certifiquei.com.br/sgbd>.

EQUALS. **Equals - Gestão Financeira Inteligente**. 2021. Acessed: 2021-02-10. Disponível em: <https://www.equals.com.br>.

EQUALS-BLOG. **Entenda o que é chargeback**. 2021. Acessed: 2021-02-25. Disponível em: <https://www.equals.com.br/chargeback-como-evitar/>.

FOUNDATION, C. N. C. **Kubernetes**. 2021. Acessed: 2021-02-26. Disponível em: <https://kubernetes.io/>.

FOUNDATION, O. **Nodejs**. 2021. Acessado: 2021-02-14. Disponível em: <https://nodejs.org/en/>.

INC., D. **What is a Container?** 2021. Acessed: 2021-02-26. Disponível em: <https://www.docker.com/resources/what-container>.

POSTGRES. **Postgres**. 2021. Acessado: 2021-02-14. Disponível em: <https://www.postgresql.org/>.

REDHAT. **O que são Microsserviços?** 2021. Acessed: 2021-04-03. Disponível em: <https://www.redhat.com/pt-br/topics/microservices/what-are-microservices>.

REDHAT. **O que é api?** 2021. Acessado: 2021-02-12. Disponível em: <https://www.redhat.com/pt-br/topics/api/what-are-application-programming-interfaces>.

REDHAT. **O que é API REST?** 2021. Acessed: 2021-04-17. Disponível em: <https://www.redhat.com/pt-br/topics/api/what-is-a-rest-api>.

REDISLAB. **Banco de dados em Memória**. 2021. Acessado: 2021-02-14. Disponível em: <https://redis.io/>.

SEVICES, A. W. **SQS Filas de mensagens**. 2021. Acessado: 2021-02-15. Disponível em: <https://aws.amazon.com/pt/sqs/>.