UNIVERSIDADE DE CAXIAS DO SUL
ÁREA DO CONHECIMENTO DE CIÊNCIAS EXATAS E ENGENHARIAS

JEAN MICHEL REBELATTO

PROPOSTA E IMPLEMENTAÇÃO DE UMA SOLUÇÃO DE INTEGRAÇÃO ENTRE SISTEMAS

BENTO GONÇALVES

2023

JEAN MICHEL REBELATTO

PROPOSTA E IMPLEMENTAÇÃO DE UMA SOLUÇÃO DE INTEGRAÇÃO ENTRE SISTEMAS

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Orientador: Profa. Dra. Helena Graziottin Ribeiro

BENTO GONÇALVES

2023

JEAN MICHEL REBELATTO

PROPOSTA E IMPLEMENTAÇÃO DE UMA SOLUÇÃO DE INTEGRAÇÃO ENTRE SISTEMAS

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Aprovado em 29/11/2023

BANCA EXAMINADORA

Profa. Dra. Helena Graziottin Ribeiro
Universidade de Caxias do Sul - UCS

Prof. Me. Alexandre Erasmo Krohn Nascimento
Universidade de Caxias do Sul - UCS

Prof. Dr. Daniel Luis Notari
Universidade de Caxias do Sul - UCS

“Não deixe sua chama se apagar com a indiferença. Nos pântanos desesperançosos do ainda, do agora não. Não permita que o herói na sua alma padeça frustrado e solitário com a vida que ele merecia, mas nunca foi capaz de alcançar. Podemos alcançar o mundo que desejamos. Ele existe. É real. É possível. É seu.”

Ayn Rand

RESUMO

A integração entre sistemas é um tema comum no dia a dia de desenvolvedores de software. À medida que mais e mais sistemas empresariais surgem, e suas demandas por eficiência aumentam, a integração de softwares emerge como um componente essencial para otimizar processos e promover a harmonia entre diversas plataformas. Esse trabalho teve por objetivo apresentar uma proposta de solução de middleware capaz de integrar informações de estoque em um sistema empresarial. Por meio de metodologias como o Iconix e auxiliado pelo método do TDD, foi possível construir um protótipo condizente com as especificações requeridas na proposta, satisfazendo os casos de uso planejados. Esse protótipo é capaz de receber dados de um sistema por meio de webhooks e transmitir esses dados para demais sistemas integrados de forma automática. Também oferece uma série de funcionalidades adicionais e de apoio ao usuário, como por exemplo, permitir realizar as configurações iniciais. O protótipo foi desenvolvido utilizando PHP, em conjunto com o framework Laravel no *backend* e Vue para o *frontend*.

Palavras-chave: Integração de software. ERP. Iconix. TDD. Laravel. Vue

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

ACID	*Atomicity, Consistency, Isolation, Durability*
API	*Application Programming Interface*
CNPJ	*Cadastro Nacional da Pessoa Jurídica*
CRM	*Customer Relationship Management*
CRUD	*Create, Read, Update, Delete*
CSS	*Cascading Style Sheets*
CSV	*Comma Separated Values*
DRY	*Don't Repeat Yourself*
ERP	*Enterprise Resource Planning*
FTP	*File Transfer Protocol*
HATEOAS	*Hypermedia as the Engine of Application State*
HTTP	*Hypertext Transfer Protocol*
IP	*Internet Protocol*
JSON	*JavaScript Object Notation*
MRP	*Material Requirement Planning*
MVC	*Model, View, Controller*
OMT	*Object Modeling Technique*
ORM	*Object-Relational Mapping*
PDV	*Ponto de venda*
PHP	*Hypertext Preprocessor*
REST	*Representational State Transfer*
RPC	*Remote Procedure Call*
RUP	*Rational Unified Process*
SAAS	*Software as a Service*

SMTP	*Simple Mail Transfer Protocol*
SOA	*Service-Oriented Architecture*
SOAP	*Simple Object Access Protocol*
UML	*Unified Modeling Language*
URI	*Uniform Resource Identifier*
WSDL	*Web Services Description Language*
XML	*Extensible Markup Language*
XP	*eXtreme Programming*
YAML	*YAML ain't markup language*

SUMÁRIO

1 INTRODUÇÃO

Nos dias atuais, com a evolução do mercado e o advento da internet e de suas tecnologias, nossa sociedade passa a ser cada vez mais digitalizada (INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATíSTICA, 2022). Cada vez mais, o acesso à informação tem se tornado fundamental para o desenvolvimento das pessoas, permitindo que elas ampliem seus horizontes, aprimorem suas habilidades e tomem decisões mais informadas em suas vidas. Além disso, a disseminação de informações precisas e confiáveis também contribui para a construção de uma sociedade mais consciente e democrática.

Segundo Turban e Volonino (2013), a informação desempenha um papel central em todos os processos e é crucial usá-la de forma estratégica. O sucesso de um negócio está diretamente ligado à capacidade da organização em gerenciar sua base de informações e aproveitar as oportunidades de diferenciação oferecidas pela tecnologia da informação (TURBAN; VOLONINO, 2013).

No ambiente empresarial a necessidade de informações precisas sempre foi importante para garantir a eficiência das operações. Esse processo é por vezes alcançado através do uso de softwares criados para centralizar diversas informações em um só local, de modo a auxiliar na tomada de boas decisões. (PATIL; MASON, 2015). Essas ferramentas tornam-se vitais para a sobrevivência de instituições em um ambiente tão competitivo como o cenário atual (RAMOS; YAMAGUCHI; COSTA, 2020).

Nas últimas décadas, a adoção de sistemas para auxiliar empresas de qualquer porte na tomada de decisões vem crescendo a cada ano. Esses sistemas são também conhecidos pela sigla ERP, *Enterprise Resource Planning*, ou Sistemas de Gestão Integrado.

Fica evidente o motivo da popularidade que os ERPs atingiram devido às vantagens que eles fornecem através de suas funcionalidades. Com o uso de um ERP, é possível gerenciar e controlar áreas como finanças, compras e vendas, estoques, produção, recursos humanos, etc. de forma integrada e automatizada. Dessa forma, o ERP proporciona maior agilidade, eficiência e transparência nos processos empresariais, além de permitir uma visão ampla e precisa do desempenho da empresa.

Existem diversas opções de sistemas ERP no mercado brasileiro. Entretanto, esses sistemas podem não compartilhar do mesmo conjunto de recursos e funcionalidades dos outros, além de possuírem limitações próprias. Também é preciso estar ciente de que existem ERPs que fornecem soluções para determinados segmentos econômicos e dessa forma não são ideais para qualquer empresa.

A necessidade de integração entre diferentes softwares para centralizar informações é um desafio crucial para as empresas modernas e para os desenvolvedores desses softwares. A

falta de integração pode resultar em perda de dados, redundância de esforços, ineficiência operacional ou retrabalho. Essa necessidade é ainda maior no ambiente empresarial em que a complexidade dos processos e a quantidade de informações geradas diariamente exigem soluções tecnológicas que possam otimizar e automatizar tarefas. O problema relacionado a integração entre aplicações surge quando esses sistemas e aplicativos utilizados pela empresa precisam trabalhar juntos e trocar informações ou recursos.

Analisando as limitações desses softwares, esse trabalho tem como meta solucionar a falta de funcionalidade de um desses sistemas, chamado Bling [1]. Esse sistema possui atualmente uma limitação relacionada ao controle de estoque de múltiplas contas. Hoje não existe funcionalidade que permita sincronizar o estoque de duas ou mais contas desse sistema entre si de forma automática. Dessa forma, não se atinge um controle centralizado dessa informação. Esse será o tema de estudo deste trabalho, bem como a criação de um prótótipo de software que possa contornar esse problema.

1.1 OBJETIVO GERAL

O objetivo principal deste trabalho é propor e desenvolver o protótipo de um *middleware* capaz de integrar e sincronizar dados de estoques entre duas ou mais contas do ERP Bling. Essa funcionalidade deve permitir aos usuários resolver a lacuna de integração de estoque entre mais de uma conta, possibilitando que os dados de estoque de uma conta sejam replicados para outras contas integradas ao sistema. Além disso, essa nova funcionalidade proporcionará aos usuários uma visão geral do estoque de todas as suas contas e a permitirá configurar a comunicação de estoque entre as contas de acordo com algumas opções disponíveis.

1.2 OBJETIVOS ESPECÍFICOS

A seguir os objetivos específicos que permitirão alcançar o objetivo geral:

- Modelar a arquitetura e os componentes do sistema.
- Definir uma solução de integração de dados que seja adequada para o usuário, que seja web e possa ser hospedada em nuvem.
- Realizar a construção de um protótipo para validar a proposta de solução.
- Criação de uma cobertura de testes das funcionalidades.

[1] www.bling.com.br

1.3 METODOLOGIA

O problema computacional que esse trabalho apresenta pertence à área de Engenharia de Software, uma subárea da Ciência da Computação que se concentra no desenvolvimento de sistemas. Para alcançar os objetivos propostos, atividades intermediárias foram planejadas.

Considerando o problema apresentado, foram feitas pesquisas e análises sobre as áreas de integração de sistemas e sobre métodos de integração existentes. Durante a segunda etapa, foram realizadas pesquisas semelhantes nos tópicos de arquiteturas de sistemas, apresentando diferentes arquiteturas atualmente utilizadas no mercado, como arquiteturas de camadas ou orientada a serviços. Essa busca foi feita em diferentes fontes de informação, como bases de dados, periódicos, livros e artigos, com o objetivo de identificar estudos relevantes para a questão de pesquisa e obter uma base científica, necessária para qualquer trabalho científico.

Após estudos sobre as diferentes formas para se realizar integração de dados, foi realizado o levantamento de requisitos para resolução do problema e a modelagem da arquitetura do sistema. Essa terceira parte do projeto, relacionada com o processo de desenvolvimento do software é baseada na aplicação da metodologia Iconix. Essa é uma metodologia ágil que se concentra na utilização de casos de uso de uma modelagem orientada a objetos para produção de software.

Em conjunto com o Iconix foi utilizado o processo chamado *Test-Driven Development* (TDD), em portugês Desenvolvimento Orientado por Testes. No TDD, é estabelecido que, antes de desenvolver cada funcionalidade, é necessário escrever testes para assegurar a qualidade do código.

Detalhes de ambas as metodologias estão descritos no capítulo 5 deste trabalho.

1.4 ORGANIZAÇÃO DO TRABALHO

Este trabalho está dividido e organizado nos seguintes capítulos

- O capítulo 2 descreve métodos de integração existentes e dificuldades comuns ao desenvolver uma integração.
- No capítulo 3 é explicado o que são ERPs. Contém uma explicação sobre o ERP Bling e descreve diferentes tipos de arquiteturas de sistemas.
- O capítulo 4 fornece detalhes sobre as metodologias e tecnologias utilizadas para o desenvolvimento da proposta.
- O capítulo 5 descreve a proposta do *middleware*, iniciando com uma visão geral da arquitetura e após descrevendo requisitos, casos de uso e diagramas de classes.

- O capítulo 6 apresenta o processo de desenvolvimento de um protótipo para validação da proposta.
- O capítulo 7 descreve as considerações finais trabalho com a conclusão e possíveis trabalhos futuros.

2 INTEGRAÇÃO DE APLICAÇÕES

Esse capítulo visa apresentar uma visão geral sobre o tema de integração de softwares. O capítulo apresenta o motivo por trás de integrações serem necessárias, dificuldades que podem surgir ao desenvolver uma integração, diferentes tipos de integração e suas vantagens e desvantagens.

2.1 A NECESSIDADE DE INTEGRAR

A integração de softwares é uma necessidade cada vez maior (RICE; SIEBEL, 2019). À medida que mais serviços estão disponíveis ao usuário, é cada vez mais desejável que esses serviços se comuniquem entre si. Com relação a empresas, à medida que adotam diferentes sistemas para gerenciar suas operações, surge o desejo de conectar esses sistemas e fazer com que eles trabalhem juntos.

A comunicação entre diferentes softwares pode ter vários objetivos. Um exemplo mais recente no caso de redes sociais seria do Facebook, que integrou a funcionalidade "reels"do Instagram dentro de sua plataforma (FACEBOOK, 2021). Reels são vídeos de curta duração, com no máximo 90 segundos. Essa integração permite que os usuários da plataforma Instagram compartilhem esses vídeos também no Facebook de forma simples e conveniente. Isso visa aumentar o engajamento e o tempo de permanência dos usuários na plataforma e conter competidores de outras redes sociais. Assim, esse tipo de integração se torna estratégico para o longo prazo de uma companhia.

Pensando na área empresarial, a necessidade de integração de aplicações também surge de diversas formas. Algumas empresas possuem diferentes sistemas para controlar várias áreas distintas do seu negócio. Integrar esses sistemas auxilia a centralizar informações em um único local. Também existem integrações que visam obter dados de um desses sistemas para fins de análise e tomada de decisões estratégicas. Além disso, a integração de aplicações também é importante para a automação de processos de negócios, permitindo que as empresas possam otimizar suas operações e aumentar sua eficiência. Com esse tipo de integração, as empresas podem melhorar sua capacidade e processos, tornando-se mais ágeis e eficazes em um mercado cada vez mais competitivo.

Dessa forma, fica clara a responsabilidade da integração em prover soluções que garantam que as informações possam ser compartilhadas de modo eficiente. Para isso, normalmente são utilizados *middlewares*, que são definidos por Linthicum em seu livro *Enterprise Application Integration* como "basicamente qualquer tipo de software que facilite a comunicação entre dois ou mais sistemas"(LINTHICUM, 2000).

2.2 POSSÍVEIS DESAFIOS

A integração de softwares prevê que dois sistemas diferentes possam se comunicar entre si, realizando a troca de informações. Isso normalmente significa lidar com sistemas de diferentes idades, tecnologias e arquiteturas, em diferentes localizações, rodando em plataformas diferentes. Existe também questões de infraestrutura, segurança e regras de negócio que devem ser levadas em consideração pois afetam o desenvolvimento de uma integração (XU, 2014).

Abaixo segue uma lista levantada de possíveis desafios que devem ser relembrados durante o processo de criar uma integração:

- Falta de conhecimento de regras de negócios: Além de entender a tecnologia por trás dos dois sistemas, para realizar uma integração é preciso entender as regras de negócio que já existem em ambas aplicações. Além disso, também pode ser necessário a criação de novas regras de negócio para definir os limites dessa integração. Dessa forma, uma integração bem-sucedida não significa apenas conseguir transferir dados de um lado para outro, mas também conectar de forma adequada diferentes unidades de regras de negócio.

- Maior escopo pode significar mais responsabilidade: Integrar diversos sistemas traz maiores implicações. Ao integrar sistemas críticos em um único local, toda a responsabilidade que antes estava distribuída em cada sistema agora é concentrada nesse único ponto. Dessa forma é vital o funcionamento correto dessa integração.

- Limitações: Um ponto importante antes de iniciarmos o desenvolvimento de uma integração é analisar a quantidade de controle que temos sobre esses sistemas. Por vezes a carência de um determinado recurso ou limites técnicos desses sistemas podem restringir a capacidade de criarmos uma integração eficiente.

- Falta de padronização: A ausência de padrões claramente definidos entre os sistemas envolvidos pode dificultar o processo de integração dos mesmos, tornando a integração mais complexa e propensa a falhas. Dessa forma é importante definir padrões claros e compatíveis para garantir uma integração bem-sucedida.

- Documentação rasa: A escassez de documentação pode tornar o trabalho de integração difícil. Sem documentação adequada, pode ser difícil entender as interfaces dos sistemas, seus requisitos e como eles se comunicam entre si.

- Segurança: Integrações podem expor ou gerar novas falhas de segurança nos sistemas envolvidos. A segurança dos dados é uma preocupação constante em qualquer processo de integração de software, e deve ser levada a sério para evitar problemas futuros.

- Monitoramento: Desenvolver um sistema de monitoramento da integração pode não ser priorizado durante a criação de uma solução. Porém, é importante monitorar e depurar a

integração para garantir que ela esteja funcionando corretamente e facilitar a identificação de qualquer problema que possa surgir.

É possível concluir que realizar uma integração pode não ser uma tarefa simples. Além dos desafios mencionados, a integração de sistemas, por ser uma parte do processo de desenvolvimento de software, traz consigo todas as outras preocupações inerentes a essa área, tais como escalabilidade, desempenho, mudanças nos requisitos, bugs introduzidos durante o desenvolvimento, entre outros.

2.3 TIPOS DE INTEGRAÇÃO

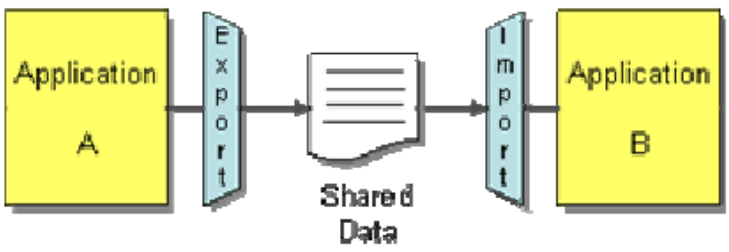
Segundo o livro *Enterprise Integration Patterns* (HOHPE; WOOLF, 2003), os tipos de integração podem ser classificados em 4 estilos principais: Integração através da troca de arquivos entre sistemas, integração através do compartilhamento de um banco de dados comum entre sistemas, integração através de chamadas de procedimentos remotos e por fim, integração através do uso de mensagens.

Embora todas as quatro abordagens resolvam essencialmente o mesmo problema, cada estilo tem suas vantagens e desvantagens. Na verdade, os aplicativos podem se integrar usando várias abordagens, de modo que cada tipo de integração aproveite a abordagem que melhor se adapta a cada parte do problema.

2.3.1 Transferência de Arquivos

Nesse tipo de integração, aplicações criam arquivos contendo as informações que são enviados para outras aplicações consumirem. A criação desse arquivo normalmente deve seguir um determinado leiaute, para ser lido pelo sistema receptor. A transmissão desses arquivos entre os sistemas pode ser feito de forma manual pelo usuário ou automatizada. O envio pode ser feito de diversas formas, como via FTP, através de e-mails ou transferência pela rede local. Os formatos de arquivo mais comuns para esse tipo de integração são XML, JSON, CSV ou texto puro. (SCEARCE, 2014)

Figura 1 – Integração por transferência de arquivos.



Fonte: (HOHPE; WOOLF, 2003)

Uma vantagem desse tipo de integração é relacionada ao nível de abstração que ela permite criar. Usando arquivos não é necessário conhecer o funcionamento interno de outra aplicação. A interface para a aplicação se torna o leiaute do arquivo que está sendo enviado ou recebido. Isso gera um alto nível de desacoplamento entre as aplicações. Cada aplicação pode efetuar alterações sem afetar as outras, desde que não esteja quebrando o contrato ao modificar o leiaute. Outra vantagem é que caso o sistema receptor esteja indisponível, o arquivo pode ser armazenado até que volte a operar.

A desvantagem desse tipo de integração é a falta de sincronismo entre as atualizações. Isso significa que as atualizações de dados podem não estar imediatamente disponíveis no sistema de destino e dessa forma limitar a capacidade de visão em tempo real desses dados.

Um exemplo de integração através de arquivos é o utilizado pelo Sistema Público de Escrituração Digital, ou Sped. Nesse caso, é gerado um arquivo de escrituração contendo informações fiscais da empresa no formato de texto. Esse arquivo segue um leiaute dividido em blocos ou também chamado de registros, onde cada registro deve conter informações pré definidas. Após a geração do arquivo é feita a validação, assinatura digital e transmissão para a base de dados da Receita Federal.

2.3.2 Compartilhamento de Banco de Dados

Na integração de banco de dados compartilhado, duas ou mais aplicações compartilham do mesmo banco de dados para consultar e armazenar informações.

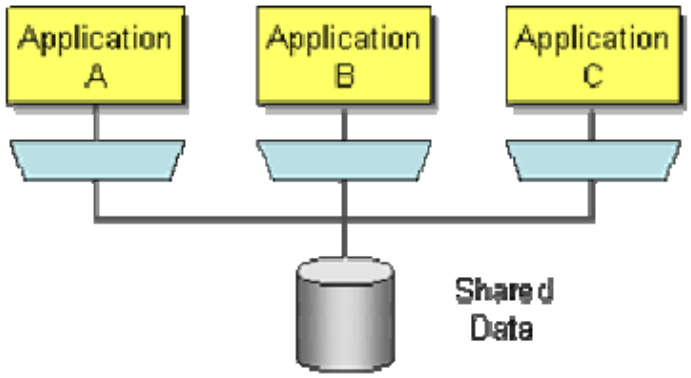
Essa estrutura de integração permite rápidas atualizações dos dados em todos os sistemas integrados. A alteração de dados em um dos sistemas atualiza a informação em todos os outros. Dessa forma a chance de inconsistências devido a falta de sincronização é menor. Esse tipo de integração também facilita a análise de dados, visto que todas as informações estão concentradas no mesmo local (STEC, 2023).

Outra vantagem dessa abordagem é a maior capacidade de combater problemas de dissonância semântica. Com um único local de armazenamento e compartilhamento de dados, as informações precisarão seguir o mesmo leiaute. Assim, não são criadas incompatibilidades no formato dos dados entre as aplicações.

Porém, apesar de apresentar algumas vantagens, esse tipo de integração tem alguns problemas que podem ser graves dependendo da situação. Conforme Fowler explica (FOWLER, 2015), criar e manter um banco de dados que permita suportar múltiplas aplicações diferentes pode gerar um *schema* mais complexo, visto que ele precisará ser mais genérico. Alterações no banco tendem a ser mais difíceis de realizar pois podem afetar diversos locais das aplicações integradas.

Também existirá uma preocupação maior relacionada a escalabilidade do banco, visto que vários aplicativos usando um banco de dados compartilhado para ler e modificar com

Figura 2 – Integração por compartilhamento de banco de dados.



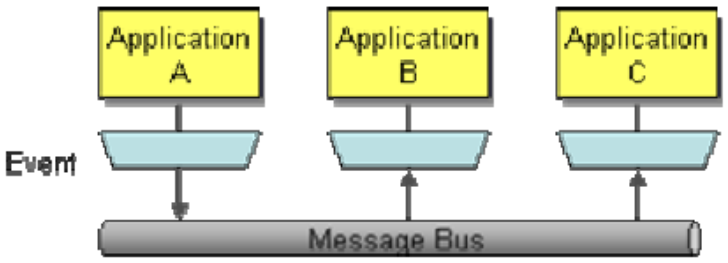
Fonte: (HOHPE; WOOLF, 2003)

frequência os mesmos dados podem causar gargalos de desempenho e comprometer a eficiência.

2.3.3 Mensagens

Nesse modelo de integração, cada aplicativo se conecta a um sistema de mensagens em comum. As mensagens são pequenos pacotes de informações que serão enviados através de um ou mais canais do sistema de mensagens, até serem recebidas pelo sistema de destino. Esse sistema de mensageria usa canais para transmitir a mensagem de forma assíncrona para o sistema alvo. As mensagens são retransmitidas até que ocorra o recebimento por parte do destinatário (FOWLER, 2015).

Figura 3 – Integração por mensagens.



Fonte: (HOHPE; WOOLF, 2003)

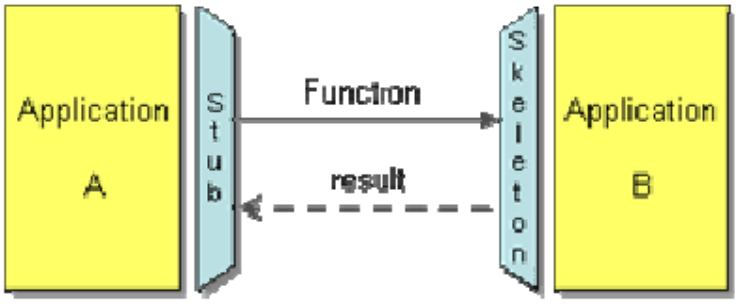
Esse tipo de integração possui algumas vantagens, como o alto nível de desacoplamento resultante em seu emprego. Por ser um modelo de integração assíncrono, o envio da mensagem não precisa aguardar a confirmação de recebimento do sistema destino. Isso ocorre pois cada pacote de dados permanece em espera no canal de mensagem para ser desenfileirada e tratada pelo sistema alvo quando for possível.

Algumas desvantagens desse tipo de integração são uma maior complexidade de desenvolvimento e a falta de sincronismo que pode ser necessário em alguns casos. Também pode ser necessário que as aplicações integradas através desse método possuam uma camada adicional de código para conseguir se comunicar com o sistema de mensageria e assim enviar e receber dados através dele (MOTA, 2022).

2.3.4 Chamadas de Procedimentos Remoto

O modelo RPC (Chamadas de Procedimentos Remoto ou em inglês *Remote Procedure Calls*) passa por um fluxo que contém dois processos: O processo de chamada e o processo servidor. O processo de chamada é feito enviando uma mensagem ao servidor com o procedimento que deseja ser executado e os parâmetros que serão utilizados. O servidor por sua vez, recebe a mensagem com a chamada do procedimento, extrai os parâmetros e executa o que for necessário. Após, é realizado o processo servidor, onde o servidor gera uma mensagem de resposta e volta a aguardar novas mensagens. As chamadas podem ser síncronas ou assíncronas, dependendo da forma como a comunicação cliente/servidor for implementada (IBM, 2023).

Figura 4 – Integração por RPC.



Fonte: (HOHPE; WOOLF, 2003)

Esse tipo de integração utiliza o princípio do encapsulamento, escondendo as estruturas de dados atrás de uma interface. Essa interface é definida pelo contrato entre o cliente e o servidor, estabelecendo os métodos disponíveis, os parâmetros necessários e os tipos de retorno esperados. O cliente realiza a chamada ao procedimento remoto através dessa interface, sem precisar se preocupar com os detalhes de implementação do servidor (FOWLER, 2015).

Esse tipo de integração tornou possível a implementação de sistemas distribuídos, que usam funcionalidades de outros sistemas.

2.3.5 Web Services

Os web services foram definidos pela W3C[1] e são padrões de integração entre aplicações com o objetivo de oferecer suporte a comunicação através de uma rede (ERL, 2009). Os web services fornecem um meio para que diferentes sistemas, tecnologias e linguagens de programação interajam e troquem dados entre si. Esses serviços podem ser disponibilizados através de uma rede utilizando diferentes protocolos.

Um web service sempre define um ou mais protocolos que utilizará para o transporte dos dados e uma interface de recursos do serviço que podem ser utilizados. Web services podem ser vistos como um avanço do RPC, pois fornecem formas mais padronizadas de comunicação entre aplicações e utilizam tecnologias de comunicação disponíveis atualmente.

2.3.5.1 SOAP

SOAP (*Simple Object Access Protocol*, em português Protocolo Simples de Acesso a Objetos) é um protocolo para a troca de mensagens. As mensagens contém os dados a serem transmitidos. Esse protocolo por sua vez emprega o uso da linguagem de marcação XML como formato de suas mensagens. Normalmente usa o protocolo HTTP ou SMTP para o envio das mensagens, mas pode funcionar com qualquer protocolo.

Esse padrão prevê o uso de um arquivo XML chamado WSDL, que tem como objetivo descrever a interface do serviço. Esse arquivo define formatos das mensagens, tipos de dados, protocolos de transporte e formatos de serialização que podem ser utilizados na comunicação. Também é utilizado para validar a estrutura de mensagens enviadas ou recebidas, verificando definições como tamanho, ocorrências e preenchimento dos elementos que compõem o arquivo (W3, 2004). Uma mensagem nesse modelo consiste em um arquivo XML composto de três partes: O envelope, que identifica o arquivo XML como uma mensagem SOAP, um cabeçalho opcional e o corpo da mensagem (W3, 2000).

Uma das principais vantagens desse tipo de integração está na ampla gama de opções para o envio das mensagens, sendo o mais comum o protocolo HTTP, mas aceitando outros como FTP e SMTP. Também é um protocolo seguro, com diversos gateways de pagamento, instituições financeiras e do governo utilizando ele em suas operações.

Desvantagens desse tipo de integração que podem ser citadas são a falta de escolha no formato dos arquivos que podem ser utilizados, visto que apenas XML é suportado. SOAP também costuma ser mais lento que outros modelos de integração e também consome mais dados devido a sua complexidade.

[1] https://www.w3.org/TR/ws-arch/#id2260892

2.3.5.2 RESTful

Outra forma de estabelecer a comunicação entre dois sistemas via web service é utilizando o padrão REST. RESTful é um termo para se referir a um web service que satisfaz todos os princípios do padrão REST (*Representational State Transfer*, em português Transferência de Estado Representacional). Doglio cita os elementos e características principais que compõem um padrão REST (DOGLIO, 2018):

- Cliente-Servidor: A existência dessa estrutura gera a separação da comunicação em duas entidades. A entidade cliente, que consome o serviço e a entidade servidor, que fornece o serviço.
- Stateless: Stateless ou sem estado significa que cada requisição é independente de outra e dados não são compartilhados entre requisições.
- Cacheabilidade: Quando aplicável, a cacheabilidade de dados bem realizada fornece um aumento significativo de desempenho para a aplicação.
- Interface Uniforme: Uma interface uniforme e padronizada é importante para facilitar a utilização do serviço por parte do cliente.

Segundo Fowler (FOWLER, 2010), para um web service ser considerado RESTful ele deve satisfazer quatro princípios que determinam o nível de maturidade de um serviço. São eles:

- Nível 0 - O web service utiliza o protocolo HTTP como sistema de transporte.
- Nível 1 - Além do nível 0, deve possuir recursos bem mapeados e representados.
- Nível 2 - Além do nível 1, os recursos devem utilizar corretamente os verbos HTTP.
- Nível 3 - Além do nível 2, deve fazer uso de HATEOAS.

O nível 0 define o uso obrigatório do protocolo HTTP, onde as chamadas e retornos ao web service devem ocorrer. Devido a natureza do HTTP, onde as requisições são independentes, nem o processo cliente e nem o servidor necessita gravar estados.

O nível 1 define o mapeamento correto dos recursos, com os nomes sendo substantivos no plural. Cada recurso é direcionado através de sua URI. Aqui já pode ser implementado o CRUD dos recursos, conforme a necessidade, utilizando os verbos HTTP como GET, POST, DELETE, UPDATE e assim por diante.

O nível 2 reforça o uso correto dos verbos HTTP. Dessa forma, pode existir apenas uma URI para o mesmo recurso, e o verbo HTTP se torna o identificador de qual operação está sendo solicitada ao web service.

O nível 3 descreve o uso de HATEOAS (*Hypermedia as the Engine of Application State* ou Hipermídia como o Mecanismo do Estado do Aplicativo). Com o uso de HATEOAS, o usuário do serviço precisa conhecer apenas a URL de acesso ao web service. Através desse link esse usuário consegue conhecer todos os outros recursos. Isso ocorre porque as requisições feitas já retornam as ligações para outros recursos do serviço.

Conforme o exemplo do Algoritmo 1, o retorno desse web service implementa o HATEOAS e permite que o usuário do serviço descubra o recurso "cursos", pois esse tem ligação com o recurso "alunos". Isso permite que o usuário necessite de pouco ou nenhum conhecimento prévio do web service e explore o serviço de forma independente (KARANAM, 2019).

Algoritmo 1 – Exemplo de retorno de um web service RESTful

```
{
    "alunos": [
        {
            "id": 1,
            "nome": "Jon Doe",
            "curso": "api.exemplo.com.br/cursos/1"
        }
    ]
}
```

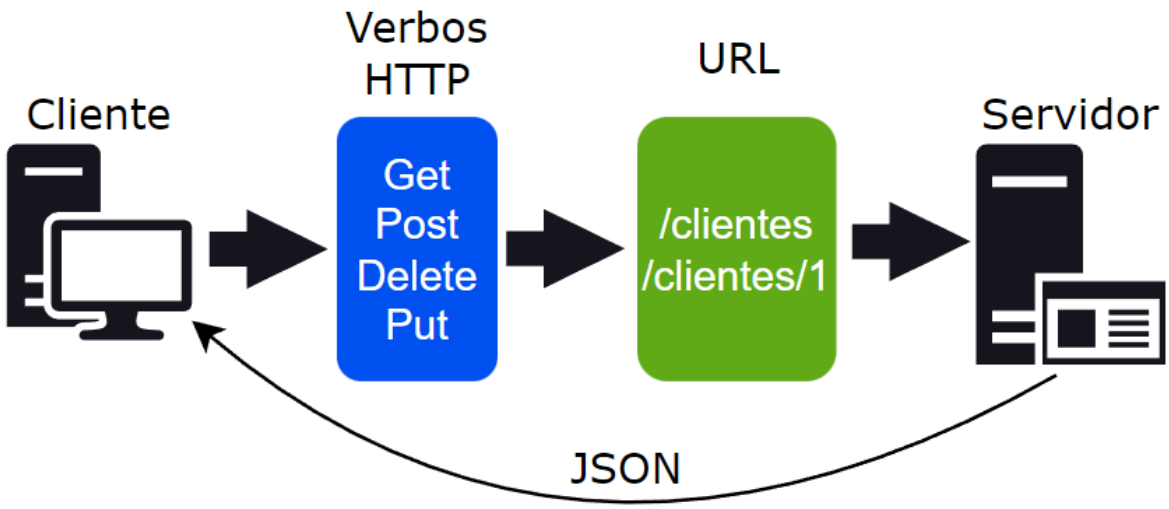
Fonte: O Autor (2023)

Um web service do tipo REST tem uma série de vantagens que impulsionaram o uso desse padrão por toda a internet. Diferente de SOAP, que obriga o uso de XML, um web service RESTful possibilita usar múltiplos formatos como texto sem formatação, HTML, XML, YAML e JSON. Ainda comparando com SOAP, o REST elimina a necessidade de envio de envelopes e de usar o XML nas mensagens enviadas, tornando a comunicação muito mais simples e leve de ser transportada (GARCIA; ABILIO, 2017).

Devido a natureza do HTTP, o REST não possui estado e cada requisição é independente. Isso permite que o servidor receba um pedido, trate ele e libere a memória utilizada, aumentando a escalabilidade. Isso também auxilia o cacheamento de dados, visto que duas requisições de verbo GET de um mesmo recurso tende a retornar a mesma resposta (ERINC, 2020).

Desvantagens de um web service RESTful que podem ser citadas é a obrigatoriedade do HTTP para transporte, sem a possibilidade de usar outros como FPT e SMTP. A falta de preservar o estado que o HTTP gera também pode ser uma desvantagem, pois determinadas informações acabam precisando ser armazenadas junto ao cliente, o que pode tornar a aplicação do lado do cliente pesada e difícil de efetuar manutenção.

Figura 5 – Integração RESTful.



Fonte: O Autor (2023)

2.3.6 Middleware

O termo *middleware* pode possuir várias definições, conforme o contexto inserido. De forma geral, se refere a um software que permite diferentes programas a trabalharem em conjunto, atuando como uma camada intermediária entre eles. Diversos softwares empregam *middlewares* para se comunicarem, como sistemas operacionais, aplicações empresariais, sistemas web entre outros.

Middlewares agem como facilitadores para obtenção de escalabilidade, manutenção e automação de um sistema. Assim, é possível particionar a carga de trabalho de maneira mais otimizada, separar melhor as entidades e dessa forma aumentar a modularidade do sistema e por fim permitir a automação de tarefas, reduzindo o risco de erros. Também é interessante ressaltar a utilidade de *middlewares* para a comunicação entre sistemas heterogêneos e que tenham uma baixa compatibilidade tecnológica entre si (GAZIS; KATSIRI, 2022).

Nesse trabalho, o termo *middleware* se refere ao sistema proposto, que tem como objetivo ser uma camada de software entre duas *APIs* do modelo REST, automatizando a comunização entre elas.

3 EVOLUÇÃO DE UM ERP

Esse capítulo explica o que é um ERP e suas principais funcionalidades, com exemplos de ERPs do mercado. Também faz uma introdução ao ERP Bling e explica o problema identificado nele que precisa de atenção. Esse capítulo também descreve diferentes tipos de arquiteturas de software utilizados, suas vantagens e desvantagens e as etapas do processo de evolução de um software.

3.1 O QUE É UM ERP

ERP (Planejamento de Recursos Empresariais, em inglês *Enterprise Resource Planning*) são softwares que tem como propósito centralizar e automatizar informações das áreas importantes dentro de uma organização, como vendas, compras, recursos humanos, estoques de produtos e matérias-primas, logística, planejamento de projetos, finanças, fiscal, contábil, de integração com sistemas externos, CRM e todas outras áreas que possam compor as operações de uma empresa (JúNIOR, 2015). A junção dos dados dessas áreas em um único local auxilia indivíduos na tomada de decisões estratégicas, pois os mesmos terão acesso a uma visão mais ampla do estado atual e futuro da empresa.

Os ERPs podem ser considerados evoluções dos sistemas MRP 2, que por sua vez evoluíram dos sistemas MRP (CORREA; CAON; GIANESI, 2007). Os sistemas MRP começaram a ser desenvolvidos no fim da década de 50, após a Segunda Guerra Mundial, para auxiliar empresas de manufatura a organizar inventários e planejar sua produção e compra de material necessário. Já o MRP 2 trouxe funcionalidades adicionais como planejamento de recursos financeiros, produtivos e de venda.

Atualmente sistemas ERP passaram a ser incorporados no processo de organizações dos mais variados setores, como serviços, comércio e revenda, distribuição e logística, saúde, financeiros e do governo.

Alguns exemplos de empresas de destaque que desenvolvem sistemas ERP são a SAP [1], Oracle [2] e Microsoft [3]. Essas empresas possuem diversos produtos de software voltados para gestão empresarial. A SAP é reconhecida por seu sistema SAP ERP, utilizado por grandes corporações para gerenciar suas operações, recursos humanos, finanças e cadeias de suprimentos. A Oracle também oferece soluções abrangentes com seu Oracle ERP Cloud, fornecendo funcionalidades avançadas de planejamento, controle e análise de negócios. Já a Microsoft possui o Microsoft Dynamics 365, uma suíte de aplicativos de negócios que inclui módulos para ge-

[1] https://www.sap.com/brazil/index.html
[2] https://www.oracle.com/br/erp/
[3] https://dynamics.microsoft.com/pt-br/

renciamento financeiro, vendas, atendimento ao cliente e operações. Essas empresas de renome continuam a aprimorar seus produtos ERP, buscando acompanhar as demandas do mercado e oferecer soluções cada vez mais completas e inovadoras.

3.2 ERP BLING

O ERP Bling [4] foi criado por um ex-professor da Universidade de Caxias do Sul, juntamente com alguns alunos na cidade de Bento Gonçalves no ano de 2008. O nome Bling é um acrônimo das palavras Bento Gonçalves e Linux.

O Bling é um sistema hospedado em nuvem, que pode ser acessado via web, com o modelo de negócio sendo o SAAS (*Software As A Service*, em português Software como Serviço). É um sistema que tem como foco principal auxiliar o dia a dia de empresas de pequeno e médio porte. Dispõe de uma gama de funcionalidades para gestão de uma empresa. Exemplos de algumas funcionalidades são: Emissão de notas fiscais, integração com diversos marketplaces, hubs e empresas de logística, cadastro de produtos e controle dos estoques, PDV e módulos financeiros.

Atualmente o sistema Bling permite que seja cadastrado uma única conta para cada CNPJ. Não é permitido inserir dados de dois CNPJs em uma mesma conta. Assim, grupos empresariais que possuam mais de um CNPJ e que utilizem o Bling como ERP precisam possuir uma conta para cada CNPJ.

Nesse modelo, empresas de um mesmo grupo que compartilhem de mesmos itens podem ter problemas com a falta de sincronismo das informações de estoque entre as contas.

Seria interessante para os usuários desse sistema possuírem uma forma de obter uma visão geral do estoque de todas as contas que pertencem a eles. Essa informação auxiliará no gerenciamento do estoque. Além disso, caso existissem formas de automatizar a comunicação do estoque entre as contas, facilitaria o uso do sistema, removendo uma operação manual que acaba sendo necessária.

Como uma grande parte desses usuários não possuem estrutura ou capital para desenvolver um software para esse problema, esse trabalho visa apresentar uma solução de *middleware* para esses usuários, de forma a contornar o problema de ausência de uma visão geral e integração de estoques de múltiplas contas.

Esse é um problema computacional da área de Ciência da Computação, cuja solução envolve elementos das área de desenvolvimento e engenharia de software e integração de dados: O ERP Bling precisa evoluir e oferecer uma nova funcionalidade de integração de dados de contas de clientes. Esse processo de integração precisa ser feito para garantir que diferentes contas possam se comunicar umas com as outras de forma eficaz e confiável, sem comprometer

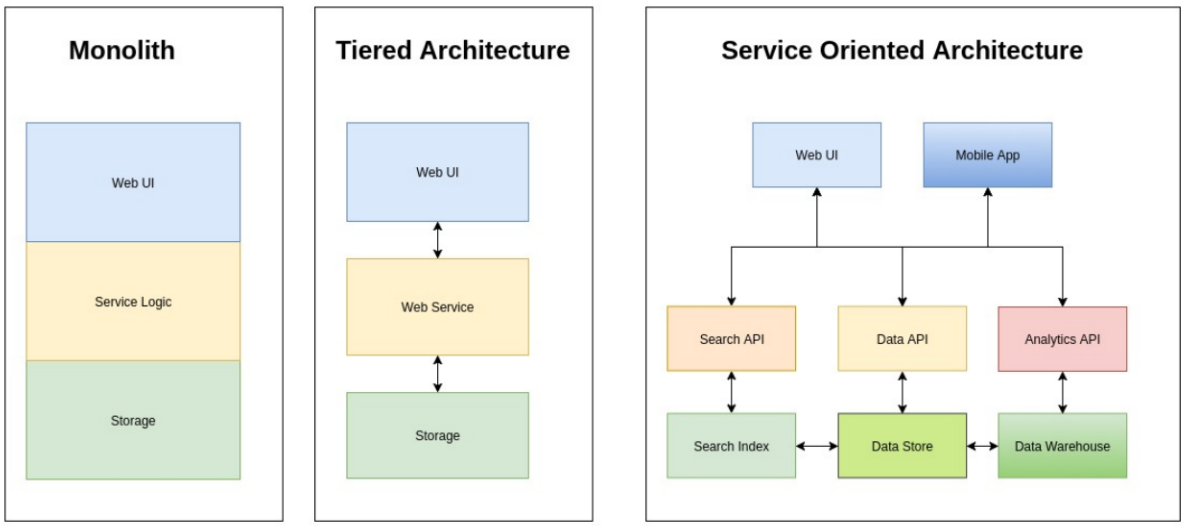
[4] https://www.bling.com.br

a integridade dos dados.

3.3 ARQUITETURAS DE SISTEMAS

Nessa seção é feito um breve estudo de diferentes arquiteturas de sistemas utilizadas atualmente. Também descreve vantagens e desvantagens no emprego de cada uma.

Figura 6 – Diferentes tipos de arquiteturas de sistemas



Fonte: Devinity, 2018

Na imagem Figura 6 são apresentadas as 3 arquiteturas que serão examinadas nesse trabalho, sendo elas as arquiteturas de sistemas monolíticos, arquitetura em camadas e arquitetura orientada a serviço.

3.3.1 Sistemas Monolíticos

A palavra monólito tem origem do grego e significa uma pedra única. No contexto de software, um sistema com uma arquitetura monolítica se refere a uma aplicação que é construída como uma unidade unificada que é autocontida e independente de outros aplicativos. Segundo Harris, "uma arquitetura monolítica é uma rede de computação grande e singular com uma base de código que une todas as preocupações de negócios"(HARRIS, 2022).

Essa é a arquitetura que tende a surgir em um software com pouco planejamento ou desenvolvido por pessoas com pouca experiência. Isso ocorre devido ao fato de ser uma arquitetura relativamente simples e conveniente nos estágios iniciais de um projeto. Porém, alguns softwares pequenos podem ser construídos de forma intencional utilizando essa arquitetura, por não haver necessidade de complexidade adicional ou por terem requisitos simples e específicos.

Algumas vantagens dessa arquitetura são a facilidade de desenvolver nos estágios iniciais e de implantação do software. Também tende a ser fácil de desenvolver testes automatizados para esse tipo de arquitetura, pois não existem dependências com outros serviços.

Porém, essa arquitetura tem uma série de potenciais desvantagens. Caso uma aplicação tenha se tornado um grande monólito, o desenvolvimento tende a se tornar lento e custoso. Pequenas alterações ficam difíceis de serem realizadas e o sistema difícil de escalar. Trabalhar em equipes maiores também se torna complexo, pois alterações de uma equipe pode impactar o trabalho de outra, devido a falta de componentes com limites bem definidos.

3.3.2 Arquitetura em Camadas

Componentes na arquitetura em camadas são organizados em camadas horizontais, com cada uma realizando algum papel específico na aplicação. Essa arquitetura não define uma quantidade específica de camadas, mas normalmente quatro camadas aparecem na maioria das aplicações, sendo: a camada de apresentação, de negócio, de persistência e do banco de dados (RICHARDS, 2015).

Nesse tipo de arquitetura uma camada vê de forma abstrata as outras camadas. Por exemplo, a camada de apresentação não precisa saber como a camada de negócio obtêm ou processa os dados. Essa camada apenas precisa receber esses dados e apresentá-los ao usuário. Assim, cada camada é independente das outras, tendo pouco ou nenhum conhecimento de como elas funcionam.

A separação das responsabilidades em nível de componente, e também em um nível maior, de cada camada, torna-se um aspecto importante dessa arquitetura, pois isso reflete numa maior facilidade em desenvolver, testar e manter aplicações.

A arquitetura de cliente-servidor pode ser considerada parte da arquitetura de camadas. O cliente é responsável por realizar as requisições ao servidor, enquanto o servidor recebe as requisições, trata elas e retorna ao cliente os recursos ou serviços necessários. Dessa forma, o lado do cliente fica encarregado da apresentação e interatividade do sistema e a parte do servidor em atender as requisições e gerenciar os dados.

Essa arquitetura é boa para uma grande variedade de sistemas, principalmente projetos que estão iniciando e ainda não tem uma visão madura dos requisitos e de qual tipo de arquitetura será necessária.

Porém, essa arquitetura não tem a melhor performance e escalabilidade devido a natureza monolítica, que cria forte dependências entre os componentes e a necessidade de passar por várias camadas para atender uma operação. O acoplamento entre os componentes e a falta de flexibilidade do código também tornam mais lento o ciclo de desenvolvimento da aplicação.

3.3.3 Arquitetura Orientada a Serviços

A arquitetura orientada a serviços (SOA - *Service-Oriented Architecture*) se baseia na criação de serviços independentes e que podem ser universalmente utilizados. Esses serviços fornecem funções de negócio que são compartilhadas.

Uma arquitetura do tipo SOA precisa primeiro de uma lista centralizada de todos os serviços disponíveis e após, cada serviço precisa descrever uma interface de forma que uma aplicação consiga realizar a comunicação conforme o contrato do serviço. Segundo Hohpe, esses são os dois elementos principais para a criação de uma arquitetura orientada a serviços (HOHPE; WOOLF, 2003).

SOA possui uma série de vantagens: Reutilização de serviços em diferentes sistemas. Permite que cada serviço utilize tecnologias e plataformas diferentes, conforme a necessidade de cada um. Um desenvolvimento e manutenção simplificados, devido aos serviços terem alto grau de desacoplamento com outras dependências. SOA também gera alto nível de abstração, com cada serviço sendo uma "caixa preta". Como cada serviço pode rodar em um servidor distinto, isso aumenta a escalabilidade da aplicação.

As desvantagens do SOA incluem aumento da sobrecarga, gerenciamento de serviço complexo e altos custos de desenvolvimento. Utilizando SOA, toda interação entre serviços requer novas validações dos parâmetros enviados, levando a tempos de resposta mais longos. Gerenciar um grande volume de mensagens trocadas entre serviços pode ser desafiador. Além disso, a implementação de SOA envolve custos significativos em termos de tecnologia e recursos humanos, devido a maior complexidade que pode apresentar.

3.4 EVOLUÇÃO DE UM SOFTWARE

A evolução de sistemas de software significa criar novos designs, porém relacionados, a partir de designs existentes. É um processo de longo prazo e tem como objetivos incluir ou expandir novas funcionalidades, fazer com que tenham melhor performance, sejam mais seguros ou integrem novas tecnologias (PRESSMAN; MAXIM, 2019).

A evolução é um processo natural da vida de uma aplicação. Independente do tamanho, objetivo ou complexidade, um software irá evoluir. A evolução de um software é necessária quando novos requisitos dos usuários são encontrados, para se adaptar a mudanças tecnológicas, demandas de mercado ou necessidades do negócio.

À medida que os usuários evoluem no conhecimento do sistema, descobrem novos usos para ele, ou novas necessidades que o software precisa suprir. As tecnologias empregadas no desenvolvimento de um aplicativo podem ser descontinuadas ou deixar de oferecer segurança, exigindo atualizações. Um software pode estar chegando próximo de seu limite técnico e a equipe responsável precisa pensar formas de melhorar seu desempenho e escalar horizontal ou

verticalmente o sistema.

A escalabilidade horizontal geralmente significa aumentar a quantidade de máquinas e consequentemente a capacidade de processamento para suportar cenários de grande volume de tráfego de forma distribuída. Já a escalabilidade vertical supõe melhorias nas especificações de hardware das máquinas já utilizadas. Ambos os conceitos tem suas vantagens e desvantagens e podem ser combinados para obter o melhor resultado.

A evolução de um software, seja através da manutenção ou na criação de novas funcionalidades tem várias semelhanças com o ciclo normal de criação de um software. Porém, existem algumas características únicas que diferem os dois processos (TRIPATHY; NAIK, 2015).

- Limitações do software existente: A evolução precisa se basear em um software que já existe. Dessa forma todo processo deve ser compatível com as limitações de código, design e arquitetura que já existam.
- Período de tempo menor: A evolução e manutenção de um software pode levar de algumas horas a até meses, enquanto o desenvolvimento de um novo software pode levar anos.
- Dados de teste disponíveis: Diferente dos casos de testes gerados do zero na criação de um software, durante o desenvolvimento de novas funcionalidades será necessário escolher testes que já existam.
- Riscos adicionais: A evolução de um software pode apresentar riscos adicionais em comparação com o desenvolvimento de um novo software. É preciso uma análise maior para diminuir impactos negativos no sistema.
- Reutilização de código: Criar novas funcionalidades em um software permite a reutilização de código. Para isso é necessário que a equipe responsável tenha um conhecimento da arquitetura do sistema a ponto de conseguir abstrair e reutilizar componentes onde for possível.

Dessa forma, a evolução de um software, seja através de manutenção ou criação de novas funcionalidades, compartilha semelhanças com o ciclo normal de desenvolvimento, mas também apresenta características únicas. A necessidade de trabalhar com as limitações do software existente demanda compatibilidade com o código, design e arquitetura preexistentes. Além disso, a evolução de software apresenta riscos adicionais, que requerem uma análise cuidadosa para mitigar impactos negativos no sistema .Esse trabalho tem o foco principal nesse tema, a evolução e incremento de funcionalidades do ERP Bling.

4 MÉTODOS E TECNOLOGIAS

As seções a seguir descrevem as metodologias e tecnologias, bibliotecas e ferramentas empregadas para criação e desenvolvimento da proposta e do protótipo do sistema.

4.1 METODOLOGIAS

Para qualquer projeto que envolva o desenvolvimento de software, é importante empregar um método bem definido para esse processo. Isso proporciona organização, eficiência e qualidade ao longo de todas as etapas do projeto, desde a análise de requisitos até a entrega final do produto.

As primeiras metodologias surgiram através de empresas que iniciaram trabalhos para definir processos que auxiliassem o desenvolvimento de software. O primeiro método recebeu o nome de modelo cascata, por Winston Royce, em 1970. Porém, o próprio Royce não recomendou o uso do método. No mesmo artigo em que Royce nomeia esse método, ele também expõe várias questões e desafios relacionados a ele. Logo após apresentar o método, ele descreve: "A implementação descrita acima é arriscada e convida ao fracasso."(ROYCE, 1970).

Com o tempo, os softwares se tornaram cada vez mais complexos e propensos a alterações e melhorias. Nesse cenário surgiu a necessidade de novos métodos que pudessem acompanhar as demandas crescentes dos usuários e das empresas. Nesse ambiente surgiram as metodologias ágeis, com foco na colaboração, na adaptação às mudanças e na entrega de valor contínua ao cliente.

4.1.1 ICONIX

O Iconix é uma metodologia criada na década de 90 por Doug Rosenberg e tinha como objetivo sintetizar e destilar os melhores aspectos de três metodologias da época que formaram o UML: Booch, OMT e Objectory (ROSENBERG; COLLINS-COPE; STEPHENS, 2005). Rosenberg explica que em vez de abranger o conteúdo das três metodologias, ele preferiu concentrar num conjunto principal de técnicas e remover a redundância e sobreposição existentes.

Assim, o Iconix utiliza um subconjunto de apenas quatro diagramas UML das metodologias anteriores, sendo:

- Diagrama de casos de uso: Apresenta as funcionalidades do sistema e as interações do usuário com o sistema.
- Diagramas de classes: Apresenta a estrutura do sistema, com suas classes, atributos, métodos e relacionamentos.

- Diagramas de sequência: Apresenta as integrações entre objetos e classes para cada caso de uso.
- Diagramas de robustez: Uma representação de como o sistema deve se comportar em resposta a uma interação do usuário.

Dessa forma, Iconix se torna uma metodologia prática, desenvolvida para ser mais ágil do que a complexidade do RUP e mais robusta comparada com a simplicidade do XP.

É uma metodologia ágil, dessa forma busca atingir algumas metas, conforme Rosemberg lista (ROSENBERG; COLLINS-COPE; STEPHENS, 2005):

- Responder a mudanças de requisitos de forma robusta e oportuna.
- Melhorar o design e a arquitetura de um projeto sem impactar massivamente seu cronograma.
- Dar aos clientes exatamente o que eles querem de um projeto conforme o que eles têm para investir.
- Realizar tudo isso sem sobrecarregar você mesmo e sua equipe.

No livro *Agile Development with ICONIX Process*, de Rosenberg, é apresentado o processo de desenvolvimento baseado em ICONIX utilizando um software de exemplo chamado Mapplet. O projeto do Mapplet é uma aplicação de mapeamento para localizar hotéis. Os passos seguidos neste livro servem de guia para as atividades de desenvolvimento planejadas nesse trabalho. Abaixo segue uma lista resumida das atividades que serão realizadas:

- Fase de levantamento de requisitos: Nesta etapa são identificados os objetos de domínio e realizados levantamentos dos requisitos. Nessa parte são identificados os requisitos necessários para o sistema cumprir o propósito ao usuário final. Assim são gerados os casos de uso e diagramas de modelo.
- Prototipação da interface: Nessa fase são gerados protótipos da interface que auxiliam o usuário e os desenvolvedores a entender ainda mais o problema e as soluções possíveis.
- Fase de arquitetura: Nessa fase é apresentada a arquitetura de alto nível, apresentando onde o *middleware* proposto nesse trabalho irá se ajustar ao sistema Bling.
- Fase de validação: Criados os diagramas de robustez. Com base nesses diagramas ocorre o refinamento do diagrama de modelo e criados os diagramas de sequência.
- Fase de implementação: O ICONIX não dá maiores detalhes dessa etapa. Nessa fase o software é desenvolvido com base nos artefatos gerados nas fases anteriores.

O uso da metodologia Iconix como referência fornece uma estrutura sólida para guiar o processo de desenvolvimento, e com flexibilidade para ajustes conforme as demandas do contexto.

4.1.2 TDD

No desenvolvimento de software, testes se referem a técnicas para validar a qualidade e funcionalidade de softwares. Eles são utilizados para garantir que o software atenda aos requisitos do usuário. Os testes são uma parte importante do processo de desenvolvimento pois permitem detectar e corrigir problemas em estágios iniciais, aumentando a qualidade e usabilidade do software final.

TDD, ou Desenvolvimento Orientado a Testes é uma técnica para desenvolver software que consiste em ciclos de desenvolvimento orientados a criar e satisfazer testes. Criado pelo mesmo autor do processo XP, o TDD se concentra no ciclo de 3 etapas: Escrever um teste, escrever código para passar no teste e refatorar o código (FREEMAN; PRYCE, 2010). Abaixo segue uma breve explicação dessas etapas:

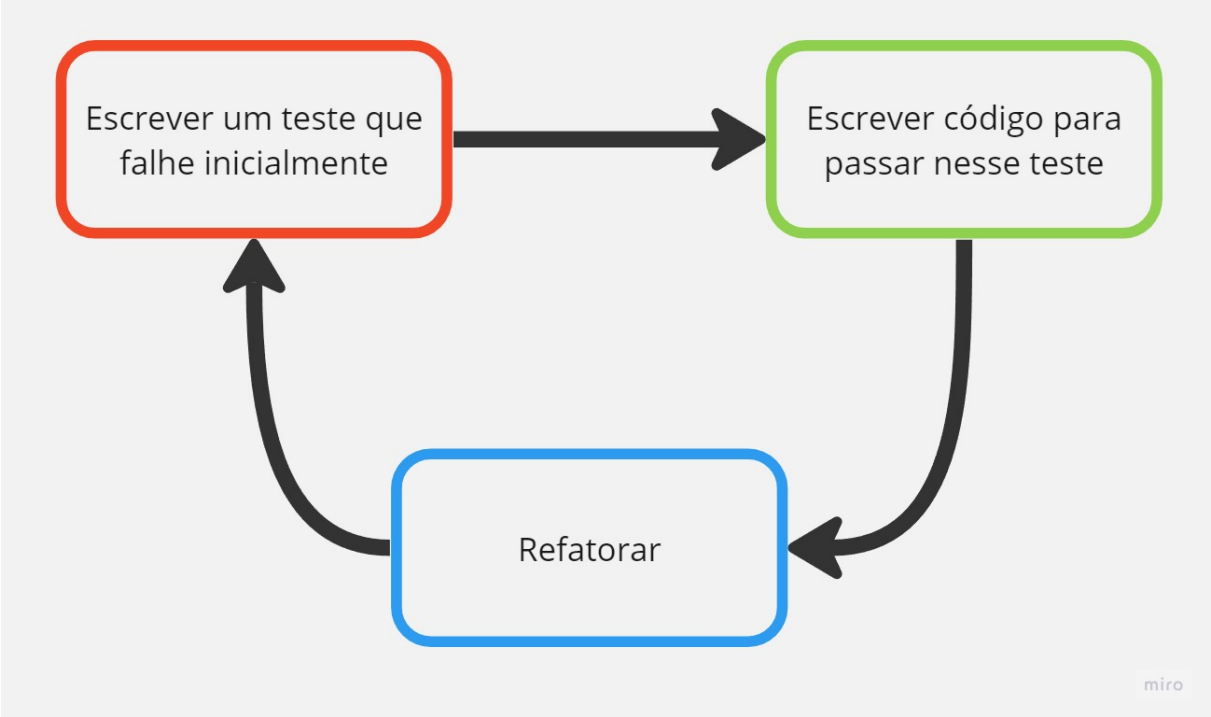
- Escrita do teste: O teste deve ser escrito antes da implementação de uma nova funcionalidade. Dessa forma, a primeira vez que o teste é rodado, ele irá falhar pela falta da funcionalidade.
- Escrita do código: O objetivo dessa etapa é escrever o código para passar no teste da forma mais rápida possível. Outros testes também devem passar para confirmar que nada foi quebrado.
- Refatoração do código: Esse é o momento em que o código é refatorado. Essa etapa tem como objetivo melhorar a estrutura e legibilidade, performance, manutenibilidade e outras características do código, sem alterar seu comportamento.

Essas etapas também são conhecidas como Vermelho-Verde-Refatorar, conforme a Figura 7 representa, com o vermelho significando falhar no teste e o verde significando passar no teste.

Para a etapa de escrita dos testes é necessário que o desenvolvedor tenha uma boa noção dos requisitos dos usuários. Isso é importante para o desenvolvedor conseguir escrever testes que condicionem o desenvolvimento de uma solução que esteja alinhada às necessidades e expectativas dos usuários finais. Dessa forma essa metodologia se associa bem com o Iconix, pois pode ajudar a garantir que o software atenda aos requisitos desejados e testado a nível de unidade (ROSENBERG; COLLINS-COPE; STEPHENS, 2005).

Outro fator que corrobora com o uso do TDD em conjunto com o Iconix é devido a cada um se concentrar em etapas distintas do desenvolvimento de um software. Enquanto o

Figura 7 – Ciclo fundamental do TDD.



Fonte: O Autor (2023)

TDD foca principalmente em testar e garantir o funcionamento do código, o ICONIX fornece uma abordagem estruturada para análise, design e modelagem dos requisitos, que é fundamental para o próprio TDD.

4.2 TECNOLOGIAS

Essa seção lista as tecnologias e ferramentas utilizadas no desenvolvimento da aplicação de integração proposta nesse trabalho.

4.2.1 PHP

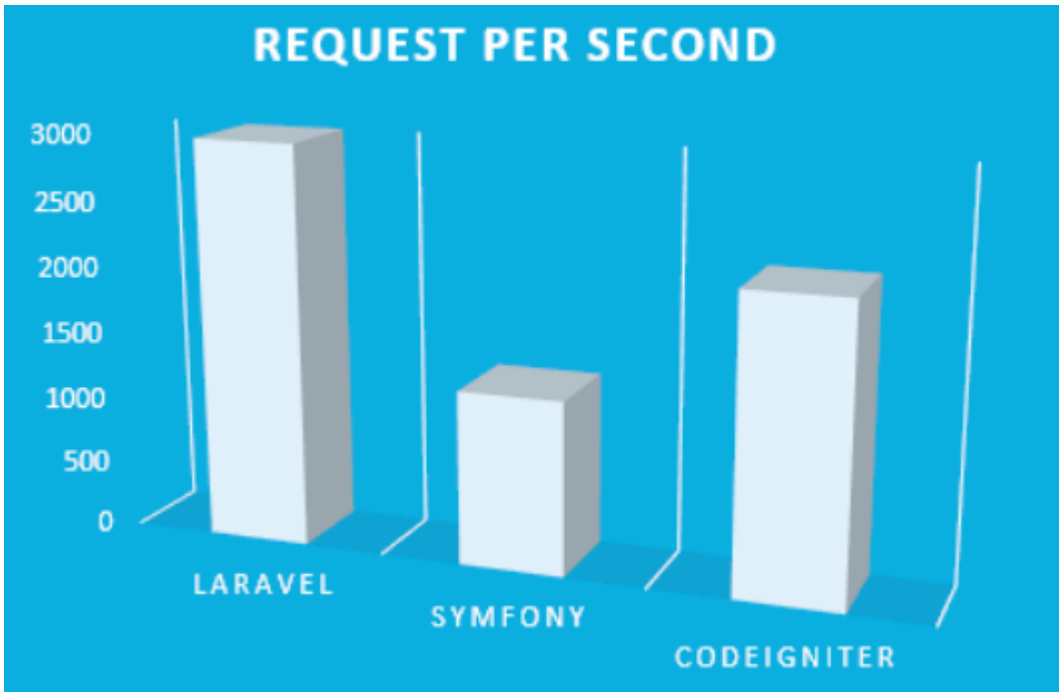
PHP[1] significa *Hypertext Preprocessor*. É uma linguagem de programação interpretada com tipagem dinâmica, criada em 1995. Atua no lado do servidor e gera conteúdos dinâmicos. Possui código aberto e não requer licença de software. Foi criada para utilizar o paradigma de programação estruturada, mas desde a versão 5 permite programar utilizando o paradigma de orientação a objetos. Segundo a pesquisa anual do *stackoverflow* de 2022 é a nona tecnologia mais utilizada por profissionais da área de desenvolvimento. Foi utilizada a versão 8.2.4 do PHP nesse trabalho.

[1] https://www.php.net

4.2.2 Laravel

Laravel[2] é um dos frameworks mais populares de PHP, tendo atualmente 73 mil estrelas e 23 mil forks no repositório do projeto no Github. Utiliza o padrão MVC (model-view-control) e possui uma sintaxe simples e sucinta. Permite ao desenvolvedor criar aplicações web de alta qualidade de forma rápida e eficiente, devido a baixa curva de aprendizado. Outro motivo de utilizar Laravel é devido a ele oferecer ferramentas que realizam tarefas comuns no dia a dia da maioria dos projetos web, como criação de rotas para a API ou para as páginas web, autenticação, tratamento de erros, ORM, testes unitários, entre outras (GARBAR, 2020). Foi utilizada a versão 10 do Laravel nesse trabalho.

Figura 8 – Requisições por segundo entre frameworks PHP.



Fonte: (LAAZIRIA *et al.*, 2019)

Além das funcionalidades que esse framework oferece, também é importante ressaltar que em comparação com outros frameworks da linguagem PHP como Symfony e CodeIgniter, o Laravel é o que apresenta a melhor capacidade de receber e lidar grandes quantidades de requisições, menor consumo de memória e o menor tempo de resposta, tornando ideal para cenários que demandam uma alta escalabilidade (LAAZIRIA *et al.*, 2019).

A Figura 8 representa a capacidade que Laravel possui em tratar grandes volumes de requisições em relação a outros frameworks PHP.

Também foi empregado o uso do Laravel Breeze, que implementa funcionalidades básicas como autenticação e autorização de usuários.

[2] https://laravel.com

4.2.3 Tailwind

Tailwind[3] é um framework CSS que fornece um conjunto de classes pré-definidas para serem utilizadas na estilização de uma interface. Possui classes para estilizar os elementos da interface, como cores, tipografia, layout e diversas outras. Permite criar interfaces personalizadas mas sem precisar criar todo CSS do zero. Foi utilizada a versão 3 dessa ferramenta no trabalho.

4.2.4 Vue

Vue[4] é um framework de JavaScript para construção de interfaces. É baseado na criação e utilização de componentes de interface, tornando mais fácil criar, reutilizar, testar e manter o código. Vue implementa reatividade nos componentes. Assim, quando ocorre uma alteração no estado desse componente, a interface do usuário é atualizada automaticamente. Foi utilizada a versão 3.3.4 do Vue no trabalho.

Para conseguir utilizar Vue junto do framework Laravel foi utilizada a biblioteca Inertia, que pode ser instalada junto do próprio Laravel.

4.2.5 MySQL

MySQL[5] é um SGBD relacional. Possui código aberto e dispõe de diversas funcionalidades como replicação de banco de dados, transações ACID (Atomicidade, Consistência, Isolamento e Durabilidade), integridade referencial, stored procedures, triggers e views, índices, chaves estrangeiras e outras. Oferece alta performance, confiabilidade e de fácil uso integrado ao PHP. Foi usada a versão 8 para esse trabalho.

4.2.6 Apache

Apache[6] é um software de servidor HTTP. Funciona como o mediador entre o servidor e o cliente. Possui código aberto e é considerado um software estável e confiável. É flexível pois permite configurar os módulos que serão ou não utilizados. É simples de configurar e tem suporte multiplataforma. Foi usada a versão 2.4 nesse trabalho.

4.2.7 Balsamiq Wireframes

Para modelar as interfaces foi utilizado o software Balsamiq Wireframes[7]. Esse software permite gerar protótipos de interfaces, onde é possível arrastar e soltar uma lista extensa

[3] https://tailwindcss.com
[4] https://vuejs.org
[5] https://www.mysql.com
[6] https://www.apache.org
[7] https://balsamiq.com

de componentes básicos que já fazem parte do programa. Esse software também possui uma comunidade que disponibiliza bibliotecas contendo diversos componentes adicionais. Foi utilizada a versão 4.2 dessa ferramenta.

5 PROPOSTA DE MIDDLEWARE

Esse capítulo apresenta a proposta de uma arquitetura para um protótipo capaz de solucionar o problema descrito, fornecendo as funcionalidades necessárias. Nessa etapa a metodologia do ICONIX é aplicada. Durante o capítulo são listados os requisitos e seus casos de uso. Com base nos casos de uso são gerados os diagramas de robustez e sequência. Esse capítulo também contém o diagrama de classes gerado a partir do diagrama de modelo e refinado a partir dos diagramas de robustez.

5.1 ARQUITETURA

O sistema proposto tem como objetivo efetuar a comunicação e sincronização de dados de estoque de duas ou mais contas do sistema Bling entre si, operando como um *middleware* desse ERP.

A arquitetura do sistema será construída com base no framework Laravel. Esse framework é baseado no padrão MVC. Conforme o livro Design Patterns (GAMMA *et al.*, 2009) explica, esse conceito define um padrão que consiste em 3 camadas: Model, View e Controller.

A camada denominada Model representa um conjunto de tabelas no banco de dados. Essa camada contém a implementação dos métodos de CRUD junto ao banco de dados. Também pode conter a lógica e implementar as regras de negócio necessárias para a aplicação. Ela é independente da camada de apresentação, e sua responsabilidade é realizar operações de leitura, atualização e exclusão de dados. Em alguns casos, essa camada também pode ser dividida em duas partes, com o Model ficando responsável apenas pelas operações junto ao banco e com uma nova camada que pode ser chamada de Service ficando responsável pelas chamadas de métodos, regras de negócio e processamento dos dados.

A camada chamada View será responsável pela apresentação dos dados ao usuário. Essa camada não precisa saber como os dados foram obtidos, apenas como deve representar eles. Além disso, a View permite ao usuário realizar interações com o sistema. Ao interagir com uma interface, o usuário estará interagindo com um controlador, da terceira camada do MVC.

A camada de Controller faz a intermediação de informações enviadas entre as camadas de Model e View. Essa camada processa os dados recebidos do usuário através da View e os envia para o Model. Da mesma forma, é o responsável por receber os dados do Model e fornecer para a View apresentá-los.

Esse projeto faz uso desse padrão devido a algumas vantagens proporcionadas, como auxiliar a separar o código em camadas de responsabilidade, aumentar o reaproveitamento de código e facilidade em manter o código limpo e organizado.

5.1.1 Integração

A comunicação do *middleware* com o Bling será feita por meio da versão 3 da API disponibilizada pelo ERP. Através dessa API serão enviadas requisições para atualizar as demais contas vinculadas ao *middleware*. O ERP também possui um *webhook* que dispara notificações de eventos sempre que ocorre mudanças no estoque de um produto da conta. Conforme a Figura 9 apresenta, dessa forma o *middleware* primeiro recebe a notificação desse evento através dos *webhooks* e por meio da API sincroniza o estoque desse produto para as demais contas.

A API do ERP fornece o padrão de arquitetura REST [1], com o envio dos dados sendo feito via HTTPS, utilizando o formato JSON. O padrão REST fornece uma interface consistente de recursos que podem ser acessados através do protocolo de forma segura. Segundo a própria documentação da API: "O REST ignora os detalhes da implementação do componente e a sintaxe de protocolo visando focar nos papéis dos componentes, nas restrições sobre sua interação e na sua interpretação de elementos de dados significativos. Ou seja, o usuário deve fazer uma requisição HTTP para algum *endpoint* disponível para solicitar, enviar ou modificar dados do sistema, então o *endpoint* de API transfere uma informação do estado do recurso ao solicitante. Essa informação é entregue via HTTP utilizando um formato de mensagem do tipo JSON."

A autenticação da API é feita utilizando o protocolo OAuth 2.0. Essa estrutura de autorização permite que aplicações obtenham acesso limitado de informações de usuários através do protocolo HTTPS, e é uma forma segura de proteger aplicações, desde que as recomendações de segurança sejam seguidas (LI, 2019). Para realizar a autenticação deve ser acessado a conta Bling, efetuar o cadastro de um aplicativo e adicionar as permissões dele. Assim será gerado o Client Id e o Client Secret que serão posteriormente utilizados para gerar os tokens de acesso da aplicação.

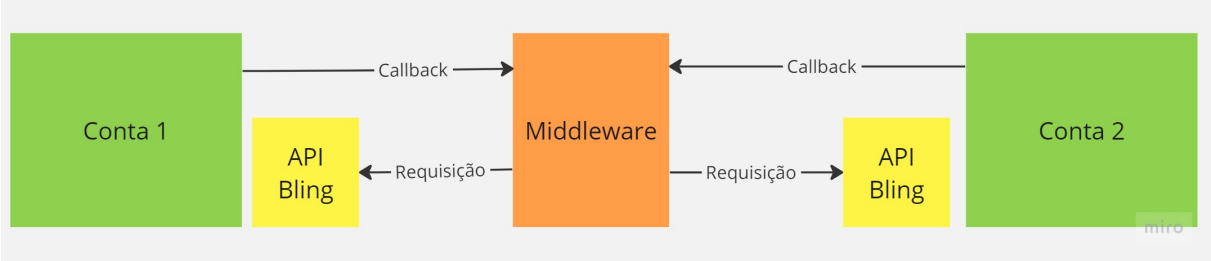
Os limites de requisições que podem ser realizadas ao utilizar recursos da API são de 10 requisições por segundo e 120 mil requisições por dia. Também existem limites quanto a quantidade de requisições com erro realizadas por um determinado IP. O limite é de 150 requisições com erros em 10 segundos, com o IP recebendo um bloqueio de 30 minutos caso o limite seja atingido. Dessa forma será necessário possuir uma forma de controlar a quantidade de requisições realizadas para que cenários de bloqueio como esse não ocorram.

A quantidade de dados retornados através de métodos GET de listagem é de 100 registros por requisição. Pode ser enviado junto da requisição um parâmetro para determinar qual página deseja ser retornada. Dessa forma, a primeira página retorna 100 registros, a segunda página outros 100 registros e assim sucessivamente.

Na Figura 10 é apresentado um fluxo que pode ser lido da esquerda para a direita. Esse fluxo representa de forma sequencial as ações realizadas por parte do ERP e do *middleware*.

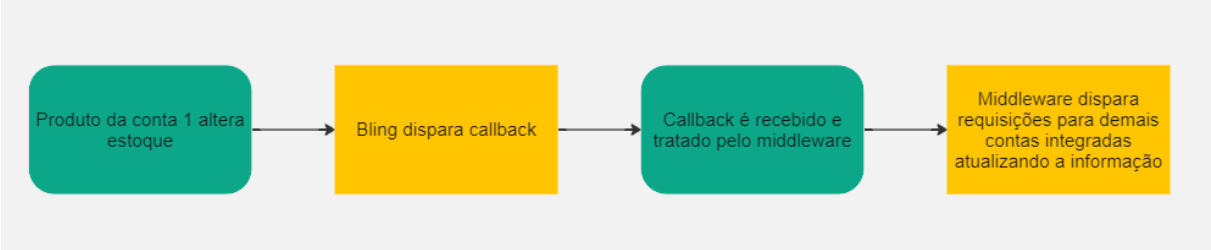
[1] https://developer.bling.com.br/bling-api#padrão-rest

Figura 9 – Diagrama representando a interação do *middleware* com o Bling



Fonte: O Autor (2023).

Figura 10 – Fluxograma do funcionamento do *middleware*

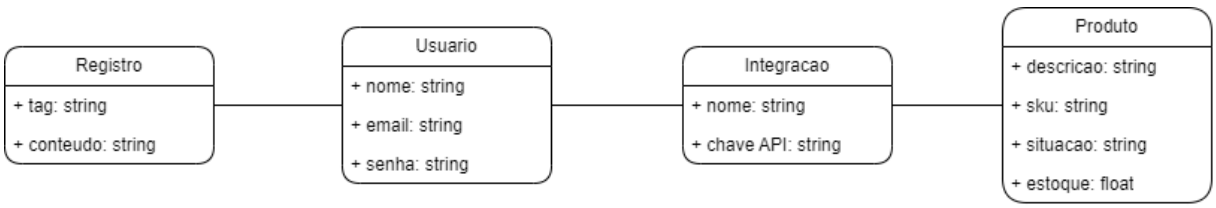


Fonte: O Autor (2023).

5.1.2 Classes

As classes fazem parte da camada de Model. Cada classe possuirá uma tabela no banco de dados. As classes possuirão os métodos de CRUD para operações no banco de dados.

Figura 11 – Diagrama de classes



Fonte: O Autor (2023).

Na Figura 11 é representado o diagrama das classes a serem implementadas no sistema. O diagrama de classes tem como objetivo apresentar de forma gráfica as classes do domínio da solução. É criado a partir da abstração dos objetos relacionados ao sistema. Os elementos desse diagrama serão as classes. Entre as classes haverão linhas que indicam relacionamentos. Esses relacionamentos identificam o tipo de relação entre duas classes.

Nessa figura a classe de usuário se refere a conta criada por um usuário, utilizando login e senha. A classe de integração representa cada integração realizada pelo usuário com uma conta do Bling. A classe de produto representa os produtos que existem no Bling e foram importados para o sistema. A classe de registro serve para manter um histórico de alterações feitas no sistema, como criação de novas integrações, importações de produtos, alterações da

situação e estoque de um produto. Será indicado nos registros se a alteração foi feita a partir de uma automação do sistema ou se foi realizada por um usuário.

5.1.3 Interfaces

As interfaces da aplicação foram pensadas para se assemelhar a interface do Bling. Isso facilitará a utilização do sistema por usuários do sistema que já estão habituados com a disposição dos elementos que pode interagir. As interfaces servirão de suporte para o usuário criar as integrações através do middleware e configurar demais parâmetros do sistema.

Foram criados mockups da interface utilizando o software Balsamiq. Dessa forma é possível contemplar a aparência e um certo grau da usabilidade da aplicação antes do desenvolvimento. Esses mockups gerados pelo Balsamiq têm uma aparência de rascunho, que ajuda a focar na estrutura e na disposição dos elementos na tela, sem se preocupar muito com os detalhes visuais. Dessa forma é possível ter uma visão geral da experiência do usuário antes de iniciar o desenvolvimento do projeto.

Os mockups das interfaces serão apresentadas juntamente com os requisitos na seção abaixo.

5.2 REQUISITOS

A primeira etapa da implementação de um software é o levantamento de requisitos. Nessa etapa os requisitos são listados e organizados. Com base nesses requisitos são gerados modelos de casos de uso e fluxos de trabalho. Esses artefatos devem ser utilizados para representar da forma mais clara possível as necessidades do usuário.

Os requisitos podem ser agrupados em dois tipos:

Requisitos de usuário, sendo descritos em linguagem natural e simples ao usuário sobre quais serviços o sistema deverá fornecer e as restrições com as quais este deve operar. A partir dos requisitos de usuários podem ser gerados os requisitos do sistema.

Requisitos de sistema, sendo descrições mais detalhadas das funções, serviços e restrições do software. O documento de requisitos do sistema define exatamente o que e como será implementado.

Além do agrupamento entre requisitos de usuários e requisitos do sistema, também é possível organizar os requisitos em funcionais e não funcionais.

5.2.1 Requisitos funcionais

Requisitos funcionais se referem a como o sistema deve se comportar. Fornecem uma descrição de quais funcionalidades o sistema deve possuir para atender determinadas necessida-

des. Esses requisitos podem descrever os resultados esperados através de uma entrada específica do usuário. Em alguns casos, também podem descrever o que o sistema não deve fazer (SOMMERVILE, 2011).

Os requisitos funcionais são geralmente derivados dos objetivos de negócios e das necessidades dos usuários e são fundamentais para validar se os objetivos alcançados pelo software criado estão alinhados com as expectativas do usuário. Esses requisitos são a base usada pelos desenvolvedores para guiar o desenvolvimento e a construção das funcionalidades do software.

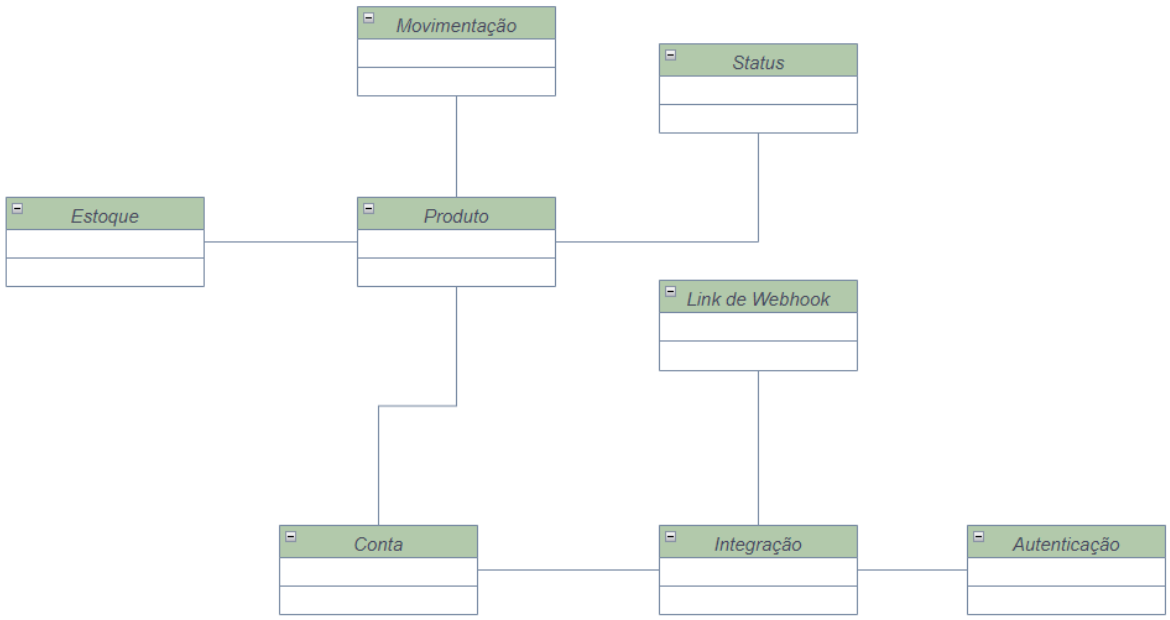
Requisitos não funcionais não se referem diretamente a comportamentos ou funcionalidades do sistema. Consideram questões como desempenho, segurança, usabilidade, confiabilidade, escalabilidade e compatibilidade. São mais críticos do que os requisitos funcionais, pois a falta de algum deles pode inutilizar completamente um projeto de software.

Os requisitos funcionais levantados são:

- R1 - O usuário deve conseguir realizar a integração com sua conta no Bling.
- R2 - O usuário deve conseguir remover a integração com sua conta no Bling.
- R3 - O usuário deve conseguir importar os produtos do Bling para o sistema.
- R4 - O usuário deve conseguir visualizar um histórico de alterações que ocorreram no sistema.
- R5 - O usuário deve conseguir exportar o histórico de alterações de estoque que ocorreram.
- R6 - O usuário deve conseguir exportar um arquivo em CSV contendo a lista de todos os produtos e seus estoques.
- R7 - O usuário deve conseguir importar um arquivo em CSV contendo produtos e seus estoques para atualizar o estoque no sistema.
- R8 - O usuário deve conseguir alterar o status de um produto no sistema para o sistema ignorá-lo e não atualizar o estoque no Bling.
- R9 - O usuário deve conseguir alterar o estoque de um produto através do sistema para atualizar as contas do Bling.
- R10 - O sistema deve atualizar o estoque dos produtos que tiverem em uma conta do Bling nas demais contas, desde que a situação do produto esteja "Ativo".
- R11 - O sistema deve gerar um link para receber o retorno do callback gerado pelo *webhook* do Bling ao ocorrer alterações no estoque de um produto.

5.2.2 Modelo de Domínio

Figura 12 – Diagrama de Domínio



Fonte: O Autor (2023).

Após possuir a lista dos requisitos é possível realizar a identificação do modelo de domínio. O modelo de domínio será um diagrama representando os principais objetos do mundo real envolvidos no sistema e as associações entre elas. Esse modelo auxilia a entender o comportamento do sistema e seus componentes. O modelo de domínio deve ser especificado sem atributos ou operações. Na Figura 12 é representado o diagrama do modelo de domínio.

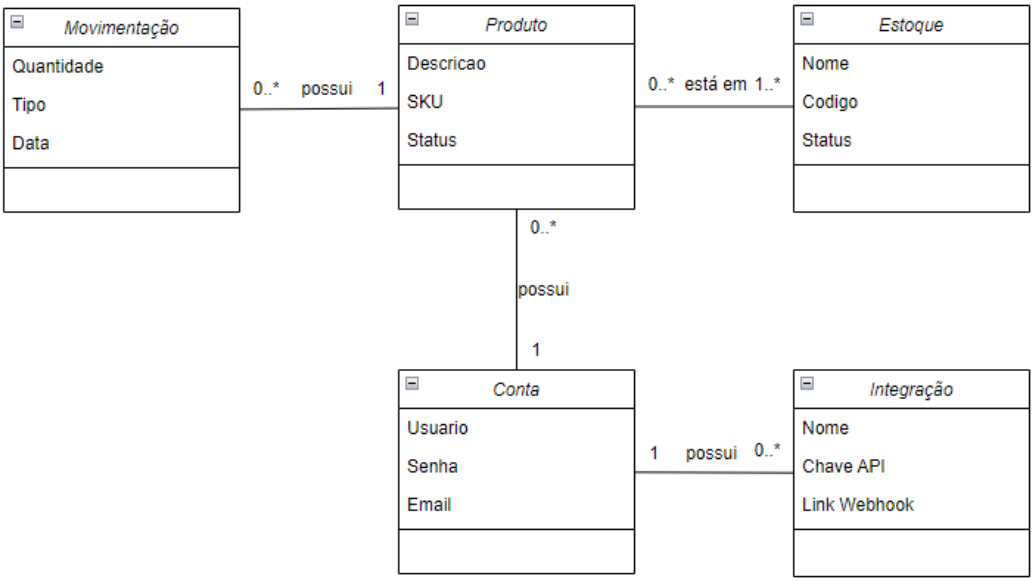
Durante o processo do Iconix esse modelo de domínio foi refinado e aprimorado, resultando na Figura 13, onde é exibido os atributos principais, a multiplicidade e a direção da relação entre os objetos.

5.2.3 Diagrama de casos de uso

O diagrama de casos de uso é um dos diagramas principais do Iconix. É um diagrama UML que apresenta as funcionalidades do sistema do ponto de vista dos atores que interagem com o sistema. O objetivo principal do diagrama de casos de uso é identificar e descrever as principais funcionalidades que o sistema deve realizar para atender às necessidades do usuário ou do negócio.

O diagrama de casos de uso é composto por dois elementos principais, os atores e os casos de uso. Os atores são os agentes externos que interagem com o sistema. Os casos de uso serão as funcionalidades do sistema que os atores poderão utilizar.

Figura 13 – Diagrama de Domínio Atualizado



Fonte: O Autor (2023).

Na Figura 14 é apresentado o diagrama de casos de uso desse projeto. Nesse diagrama é possível observar os atores usuário e sistema. O ator usuário se refere ao agente que utilizará a aplicação para efetuar o controle de estoque de suas contas do sistema Bling. Já o ator sistema se refere ao próprio middleware, conforme as operações automáticas realizadas por ele.

5.2.4 Requisito R1 - Criar integração

O usuário deve conseguir realizar a integração com sua conta no Bling: O usuário deve conseguir acessar o sistema, acessar a aba de integrações e criar uma nova integração com uma de suas contas no Bling. Será aberto um modal para preenchimento da descrição e chave API. A chave API é necessária para a integração funcionar. Essa chave deve ser obtida no sistema Bling.

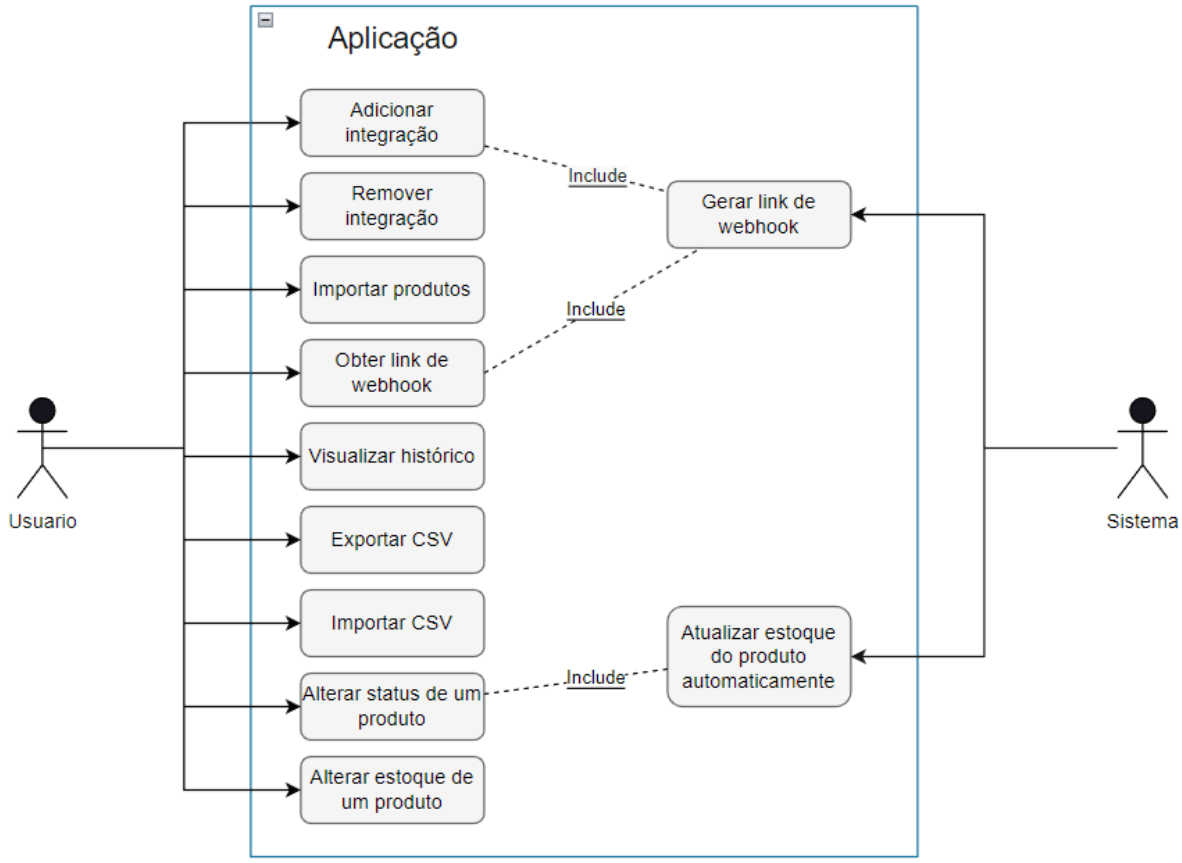
5.2.4.1 Protótipo da interface

Na Figura 15 é apresentado o protótipo de interface com o modal que surge ao clicar no botão "+ Nova integração". Nesse modal o usuário deve inserir uma descrição para a integração e a chave API obtida no Bling.

5.2.4.2 Diagrama de Robustez

Na Figura 16 é apresentado o diagrama de robustez do caso de uso 1 - Criar integração.

Figura 14 – Diagrama de casos de uso



Fonte: O Autor (2023).

5.2.4.3 Diagrama de Sequência

Na Figura 17 é apresentado o diagrama de sequência do caso de uso 1 - Criar integração.

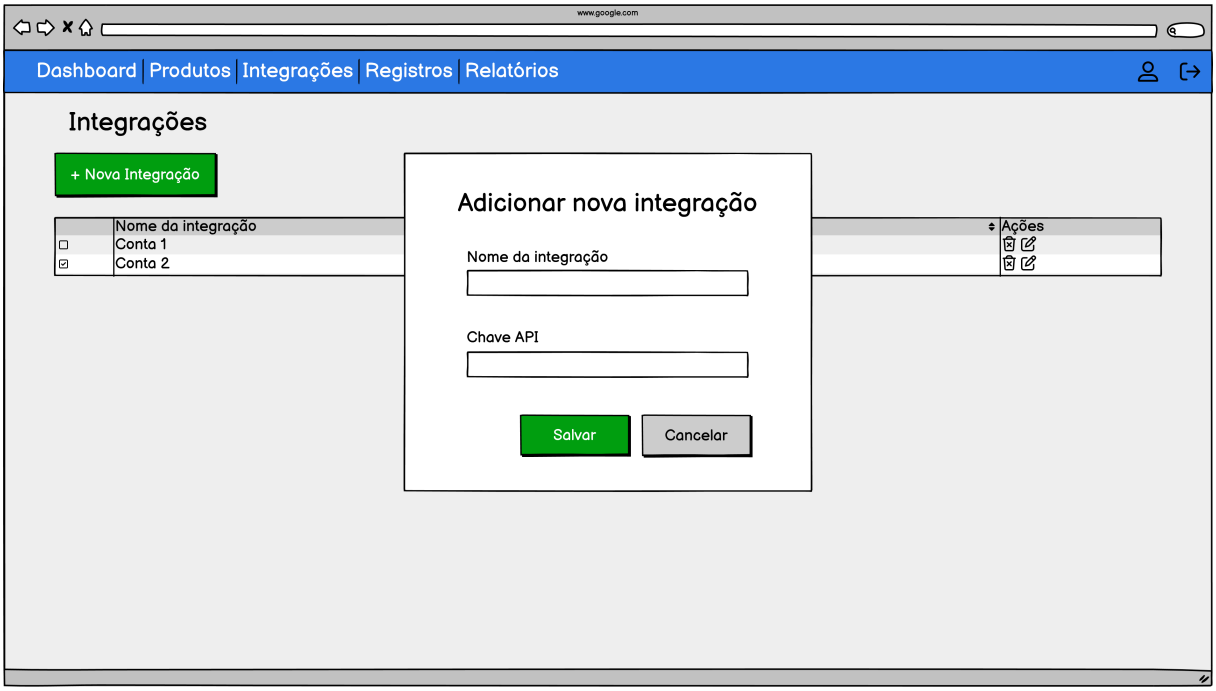
5.2.4.4 Cenários

Cenário de sucesso:

- 1 - Usuário acessa o sistema.
- 2 - Usuário acessa a aba Integrações.
- 3 - Usuário clica no botão "+ Nova Integração".
- 4 - Usuário preenche a descrição e chave API.
- 5 - Usuário clica em Salvar.
- 6 - A integração é incluída na listagem.

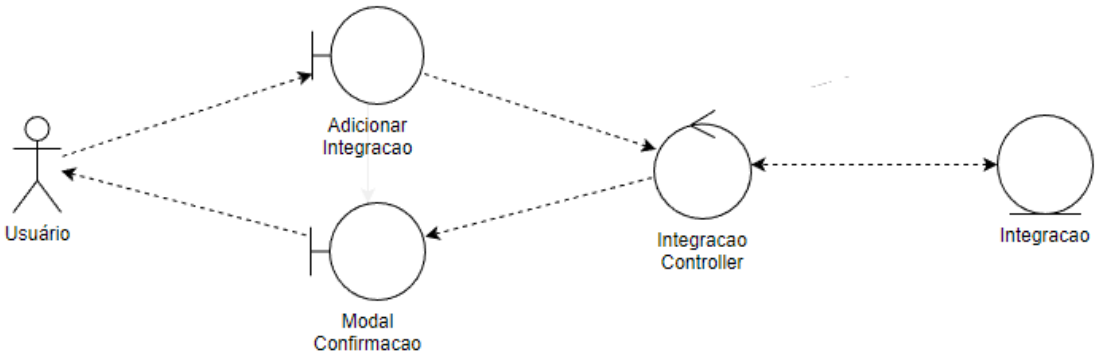
Fluxo alternativo 1:

Figura 15 – Requisito R1 - Criar integração



Fonte: O Autor (2023).

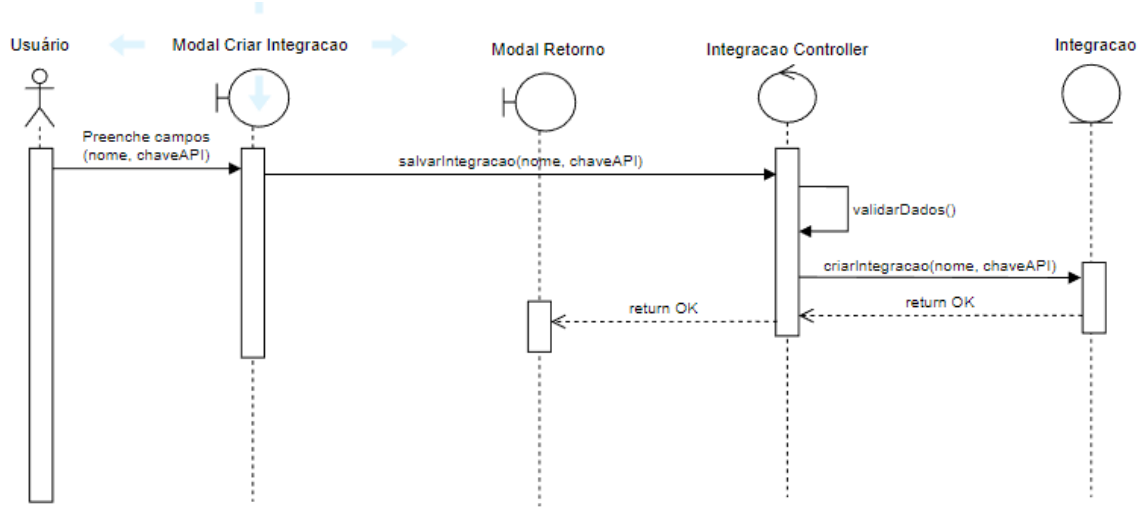
Figura 16 – Diagrama de Robustez 1 - Criar integração



Fonte: O Autor (2023).

- 1 - Usuário acessa o sistema.
- 2 - Usuário acessa a aba Integrações.
- 3 - Usuário clica no botão "+ Nova Integração".
- 4 - Usuário não preenche a descrição ou chave API.
- 5 - Usuário clica em Salvar.
- 6 - Sistema retorna erro informando a necessidade de preencher todos os campos.

Figura 17 – Diagrama de Sequência 1 - Criar integração



Fonte: O Autor (2023).

5.2.5 Requisito R2 - Excluir integração

O usuário deve conseguir excluir a integração com sua conta no Bling: O usuário deve conseguir acessar o sistema, acessar a aba de integrações e clicar no ícone de lixeira ao lado da integração que deseja remover de sua conta.

5.2.5.1 Protótipo da interface

Na Figura 18 é apresentado o protótipo de interface com o modal que surge ao clicar no ícone de lixeira ao lado da integração que deseja excluir. Surge o modal para confirmar a exclusão.

5.2.5.2 Diagrama de Robustez

Na Figura 19 é apresentado o diagrama de robustez do caso de uso 2 - Criar integração.

5.2.5.3 Diagrama de Sequência

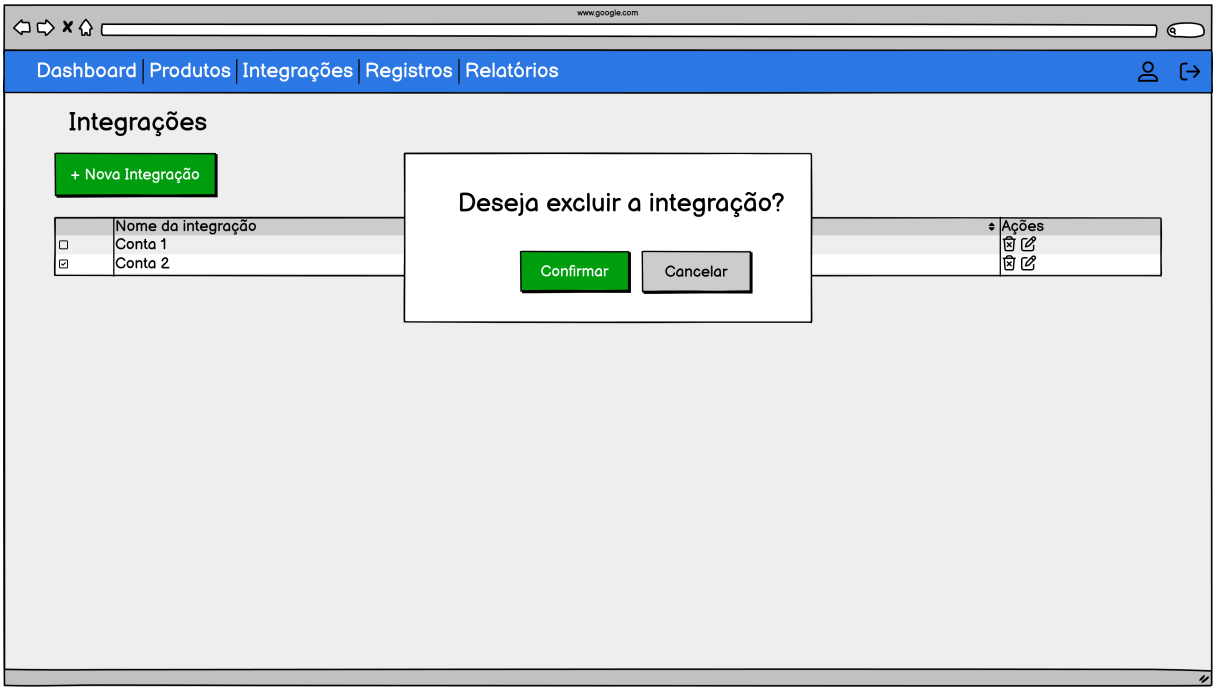
Na Figura 20 é apresentado o diagrama de sequência do caso de uso 2 - Criar integração.

5.2.5.4 Cenários

Cenário de sucesso:

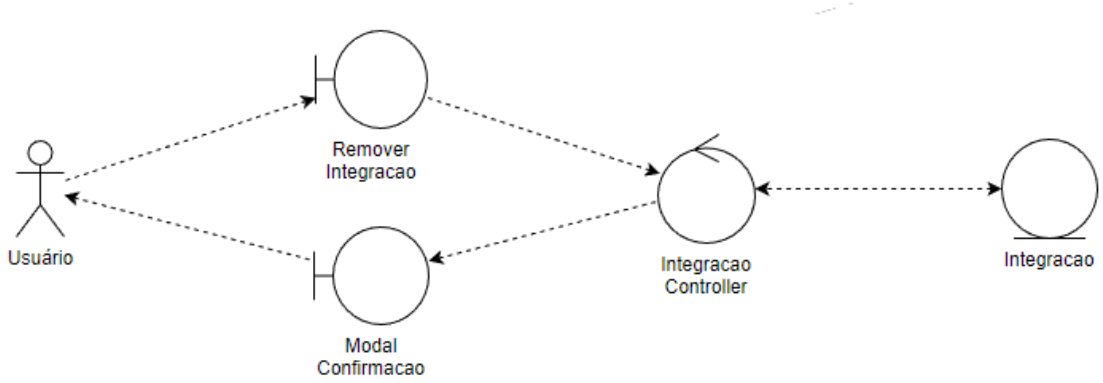
- 1 - Usuário acessa o sistema.
- 2 - Usuário acessa a aba Integrações.

Figura 18 – Requisito R2 - Excluir integração



Fonte: O Autor (2023).

Figura 19 – Diagrama de Robustez 2 - Excluir integração



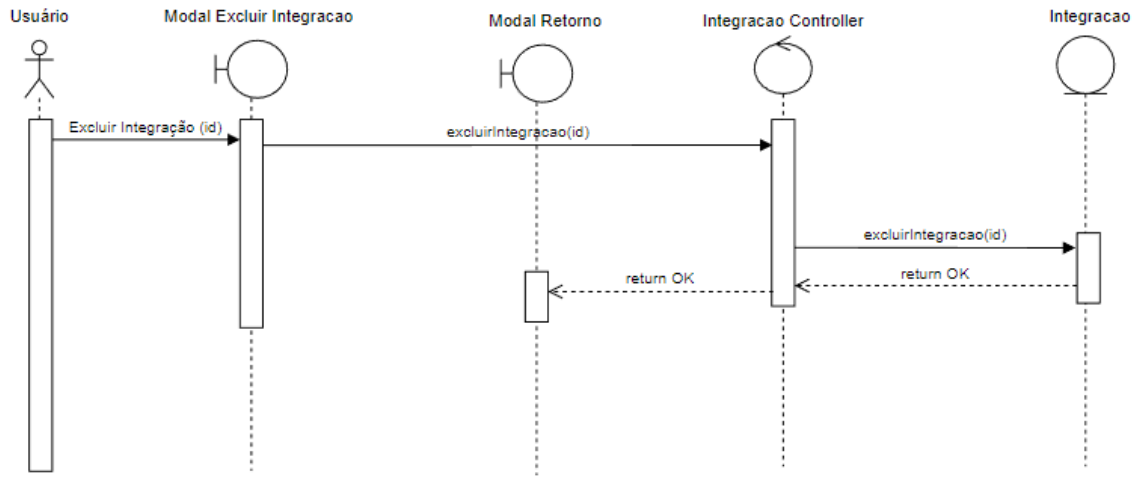
Fonte: O Autor (2023).

- 3 - Usuário clica no ícone de lixeira ao lado da integração.
- 4 - Usuário confirma a exclusão no modal que surje.
- 5 - A integração é removida da listagem.

Fluxo alternativo 1:

- 1 - Usuário acessa o sistema.
- 2 - Usuário acessa a aba Integrações.

Figura 20 – Diagrama de Sequência 2 - Criar integração



Fonte: O Autor (2023).

- 3 - Usuário clica no ícone de lixeira ao lado da integração.
- 4 - Usuário não confirma a exclusão no modal que surje.
- 6 - Modal fecha e nenhuma alteração nas integrações ocorre.

5.2.6 Requisito R3 - Importar produtos

O usuário deve conseguir importar os produtos do Bling para o sistema: O usuário deve conseguir acessar o sistema, acessar a aba de produtos e clicar no botão "Importar produtos". Isso abrirá um modal que permitirá selecionar qual integração deseja importar os produtos e confirmar a operação.

5.2.6.1 Protótipo da interface

Na Figura 21 é apresentado o protótipo de interface com o modal que surge ao clicar no botão "Importar produtos", na tela de produtos. Surge o modal para confirmar a importação dos produtos.

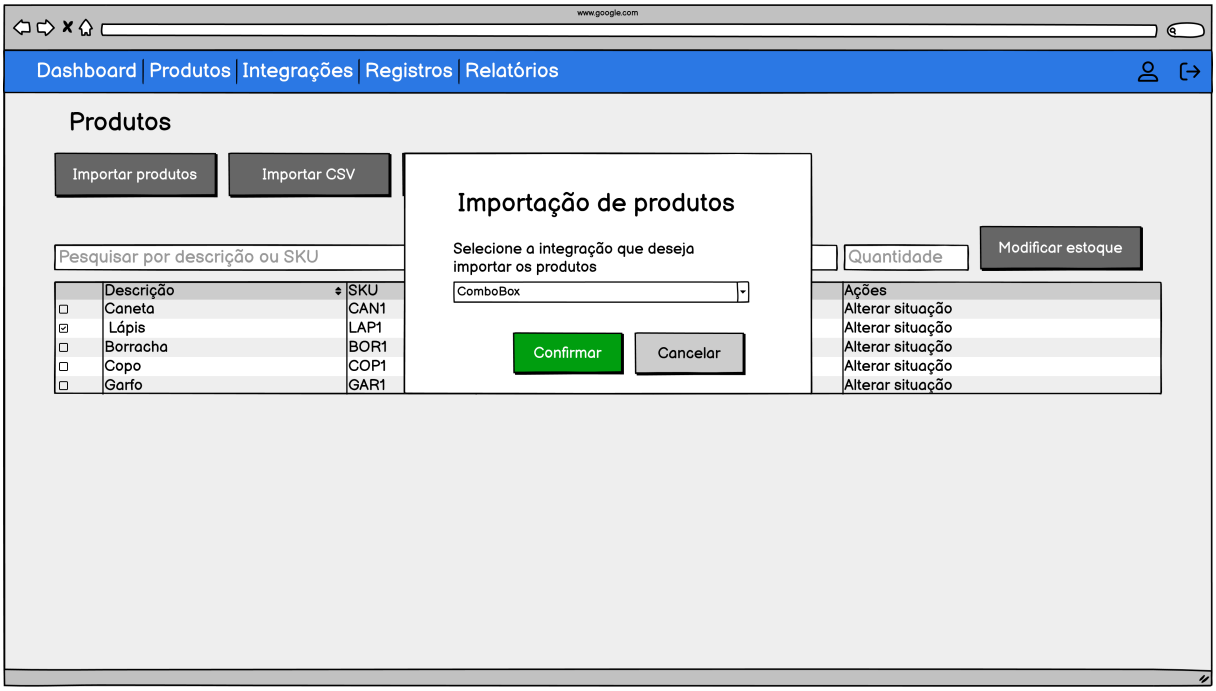
5.2.6.2 Diagrama de Robustez

Na Figura 22 é apresentado o diagrama de robustez do caso de uso 3 - Importar produtos.

5.2.6.3 Diagrama de Sequência

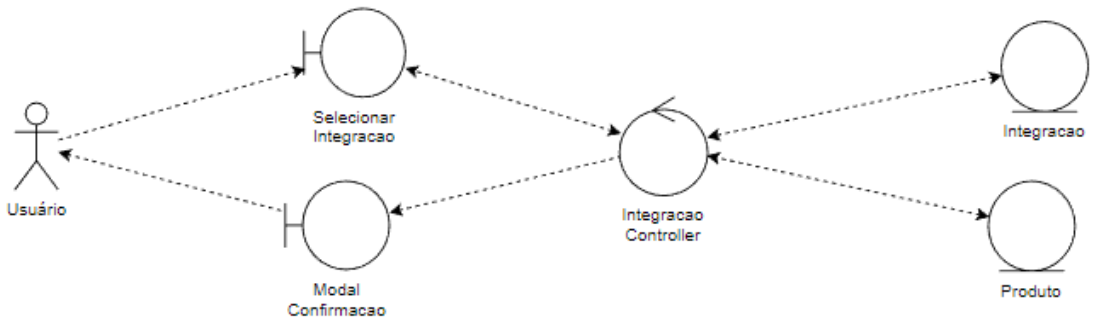
Na Figura 23 é apresentado o diagrama de sequência do caso de uso 3 - Importar produtos.

Figura 21 – Requisito R3 - Importar produtos



Fonte: O Autor (2023).

Figura 22 – Diagrama de Robustez 3 - Importar produtos



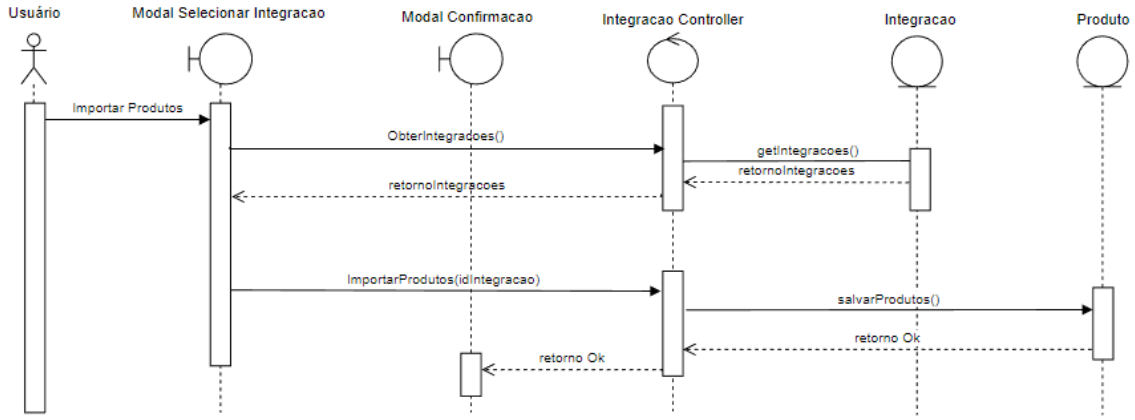
Fonte: O Autor (2023).

5.2.6.4 Cenários

Cenário de sucesso:

- 1 - Usuário acessa o sistema.
- 2 - Usuário acessa a aba Produtos.
- 3 - Usuário clica no botão "Importar produtos".
- 4 - Usuário seleciona a integração que deseja importar os produtos e clica para confirmar a operação.

Figura 23 – Diagrama de Sequência 3 - Importar produtos



Fonte: O Autor (2023).

- 5 - Os produtos dessa integração são importados para a conta.

Fluxo alternativo 1:

- 1 - Usuário acessa o sistema.
- 2 - Usuário acessa a aba Produtos.
- 3 - Usuário clica no botão "Importar produtos".
- 4 - Usuário não possui integração configurada em sua conta.
- 5 - O sistema apresenta uma mensagem informando que é preciso possuir uma integração.
- 6 - Modal fecha e nenhuma alteração nas integrações ocorre.

Fluxo alternativo 2:

- 1 - Usuário acessa o sistema.
- 2 - Usuário acessa a aba Produtos.
- 3 - Usuário clica no botão "Importar produtos".
- 4 - Modal aparece e usuário não confirma a operação.
- 5 - Modal fecha e nenhuma alteração nas integrações ocorre.

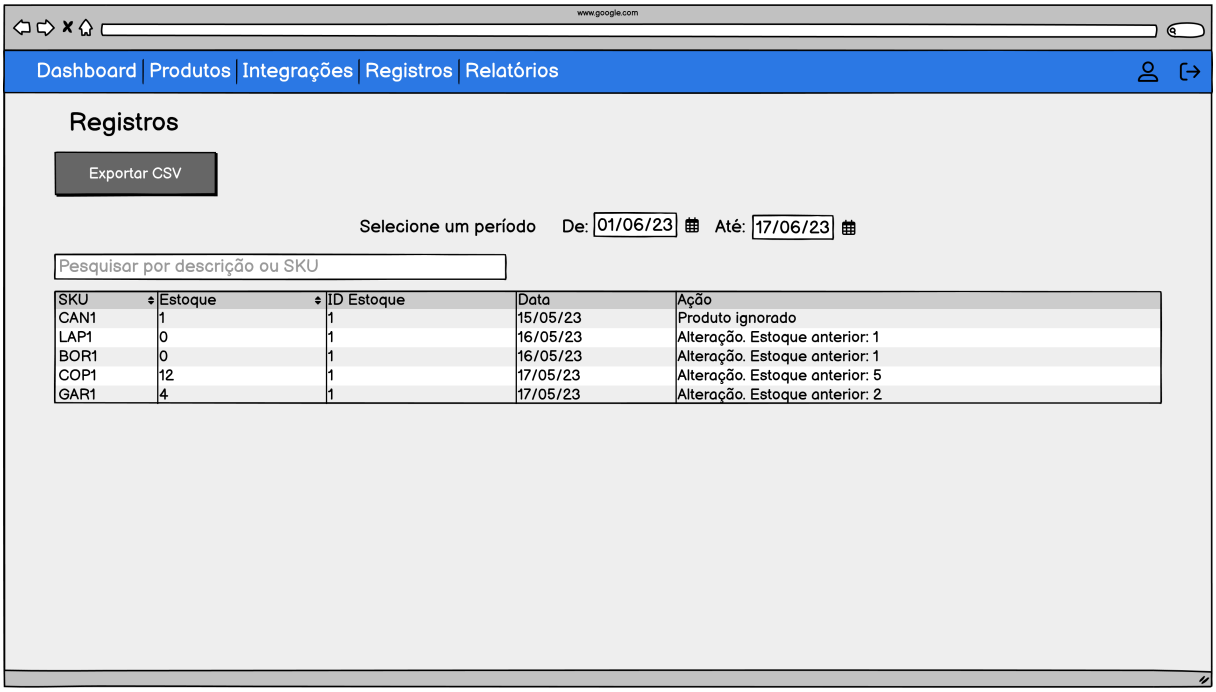
5.2.7 Requisito R4 - Visualizar histórico

O usuário deve conseguir visualizar um histórico de alterações que ocorreram no sistema: O usuário deve conseguir acessar o sistema e acessar a aba Registro. Nesse local deve ser apresentado o histórico de alterações que ocorreram. Essas alterações podem ser mudanças na situação ou estoque de um produto, novas integrações criadas e importações de produtos.

5.2.7.1 Protótipo da interface

Na Figura 24 é apresentado o protótipo de interface. Nessa interface serão listadas todas as alterações de estoque que ocorreram com os produtos.

Figura 24 – Requisito R4 - Visualizar histórico



Fonte: O Autor (2023).

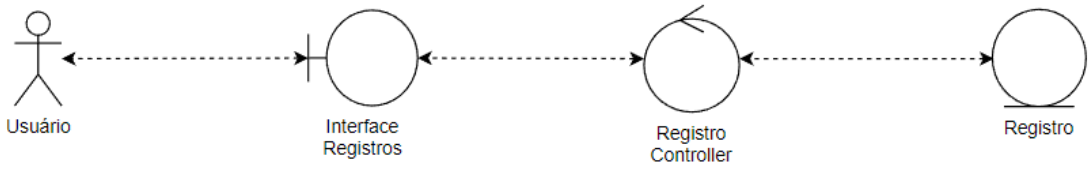
5.2.7.2 Diagrama de Robustez

Na Figura 25 é apresentado o diagrama de robustez do caso de uso 4 - Visualizar histórico.

5.2.7.3 Diagrama de Sequência

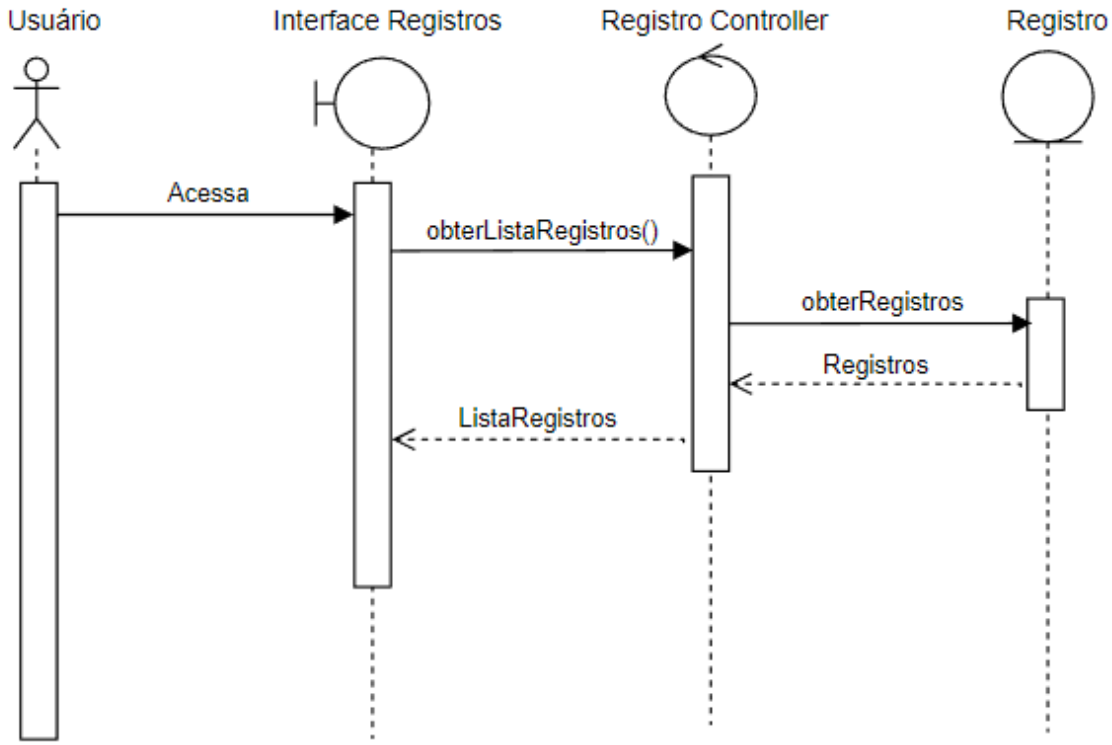
Na Figura 26 é apresentado o diagrama de sequência do caso de uso 4 - Visualizar histórico.

Figura 25 – Diagrama de Robustez 4 - Visualizar histórico



Fonte: O Autor (2023).

Figura 26 – Diagrama de Sequência 4 - Visualizar histórico



Fonte: O Autor (2023).

5.2.7.4 Cenários

Cenário de sucesso:

- 1 - Usuário acessa o sistema.
- 2 - Usuário acessa a aba Registros.
- 3 - Sistema apresenta uma lista com todas os registros.

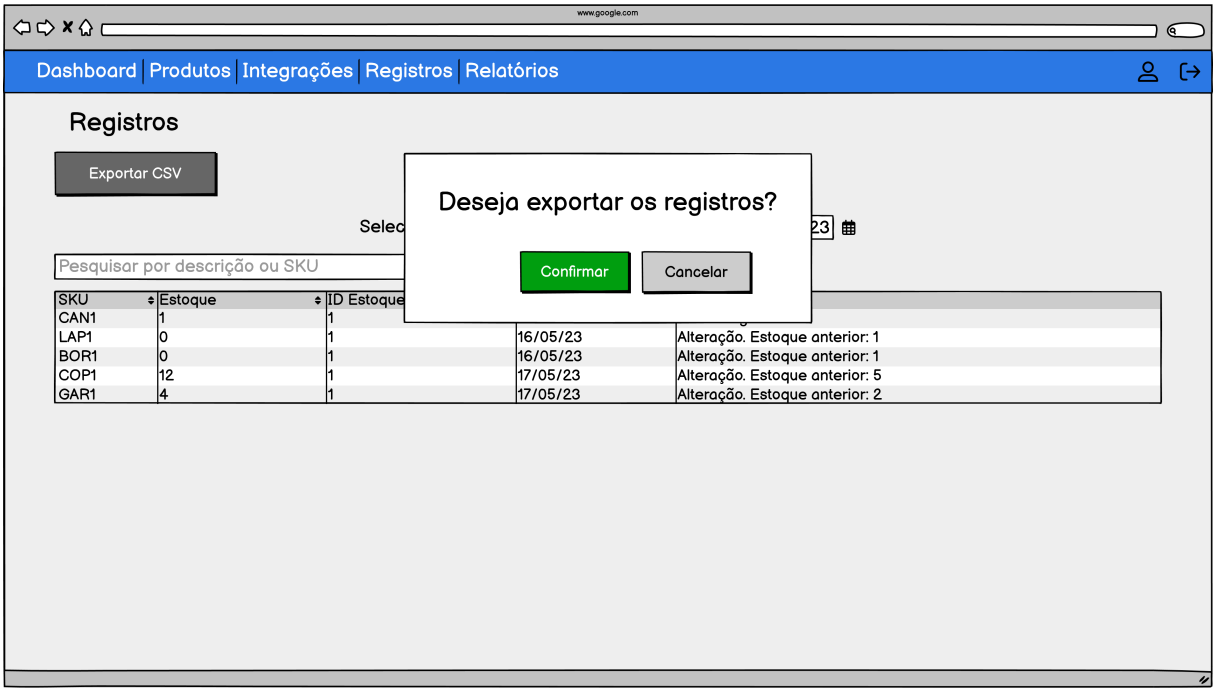
5.2.8 Requisito R5 - Exportar planilha de histórico

O usuário deve conseguir exportar o histórico de alterações de estoque que ocorreram.: O usuário deve conseguir acessar o sistema, acessar a aba Histórico. Nesse local o usuário pode exportar um arquivo CSV contendo as alterações.

5.2.8.1 Protótipo da interface

Na Figura 27 é apresentado o protótipo de interface. Essa interface apresenta o modal que será apresentado para confirmar a exportação do arquivo CSV contendo o histórico de alterações.

Figura 27 – Requisito R5 - Exportar planilha de histórico



Fonte: O Autor (2023).

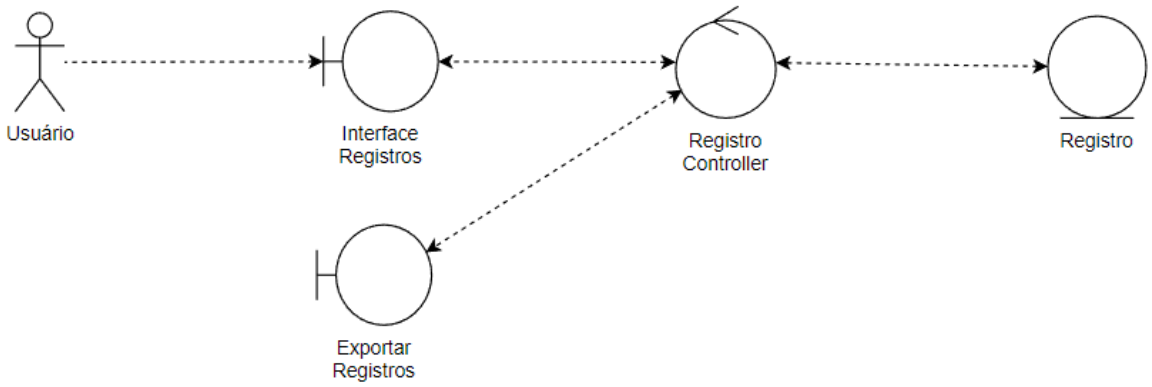
5.2.8.2 Diagrama de Robustez

Na Figura 28 é apresentado o diagrama de robustez do caso de uso 5 - Exportar planilha de histórico.

5.2.8.3 Diagrama de Sequência

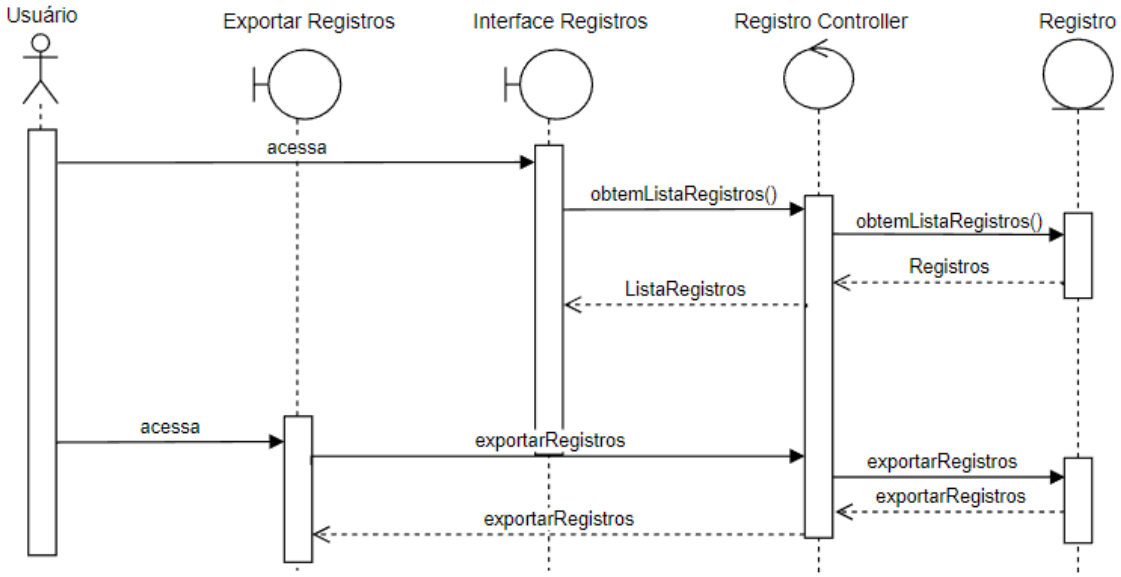
Na Figura 29 é apresentado o diagrama de sequência do caso de uso 5 - Exportar planilha de histórico.

Figura 28 – Diagrama de Robustez 5 - Exportar planilha de histórico



Fonte: O Autor (2023).

Figura 29 – Diagrama de Sequência 5 - Exportar planilha de histórico



Fonte: O Autor (2023).

5.2.8.4 Cenários

Cenário de sucesso:

- 1 - Usuário acessa o sistema.
- 2 - Usuário acessa a aba Registros.
- 3 - Usuário clica em Exportar CSV e faz o download do arquivo contendo os registros.

Fluxo alternativo 1:

- 1 - Usuário acessa o sistema.
- 2 - Usuário acessa a aba Registros.
- 3 - Usuário clica em Exportar CSV e o sistema apresenta mensagem informando que não existem registros para serem exportados.

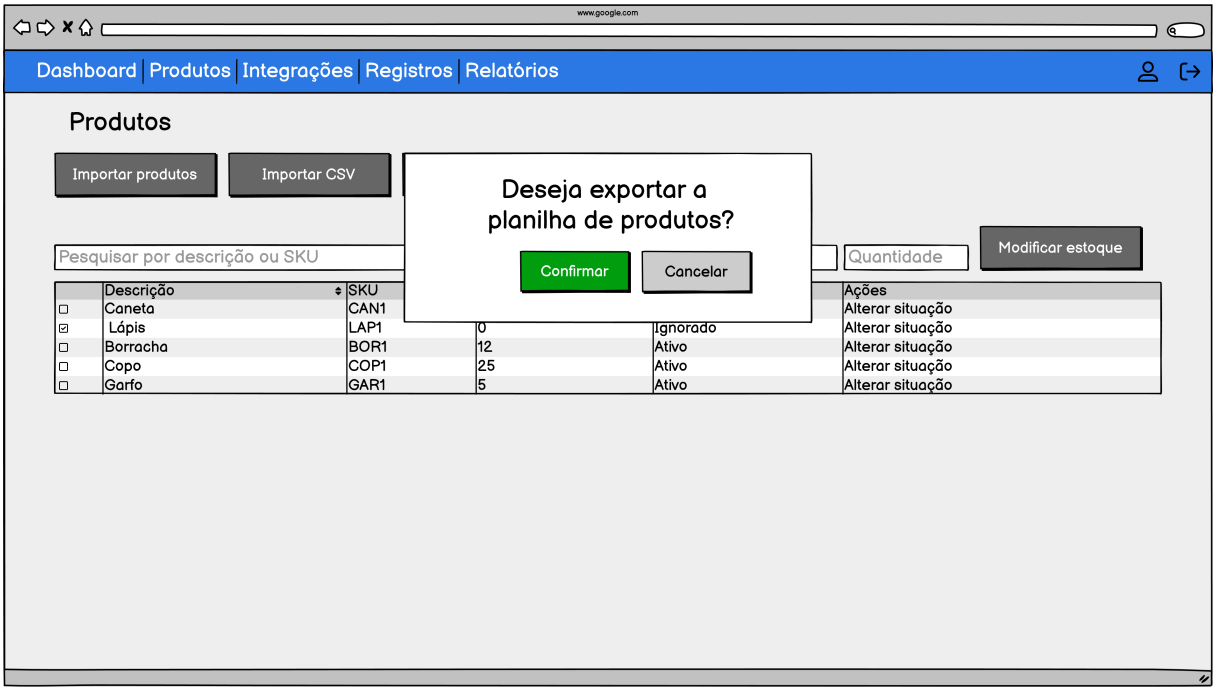
5.2.9 Requisito R6 - Exportar planilha de produtos

O usuário deve conseguir exportar um arquivo em CSV contendo a lista de todos os produtos e seus estoques: O usuário deve conseguir acessar o sistema, acessar a aba Produtos. Nesse local ele clica no botão Exportar CSV. Será apresentado um modal para confirmar a exportação. Caso existam produtos cadastrados, será gerado um arquivo CSV com os dados desses produtos.

5.2.9.1 Protótipo da interface

Na Figura 30 é apresentado o protótipo de interface. A imagem apresenta o modal que será apresentado para confirmar a exportação do arquivo CSV contendo o histórico de alterações.

Figura 30 – Requisito R6 - Exportar planilha de produtos

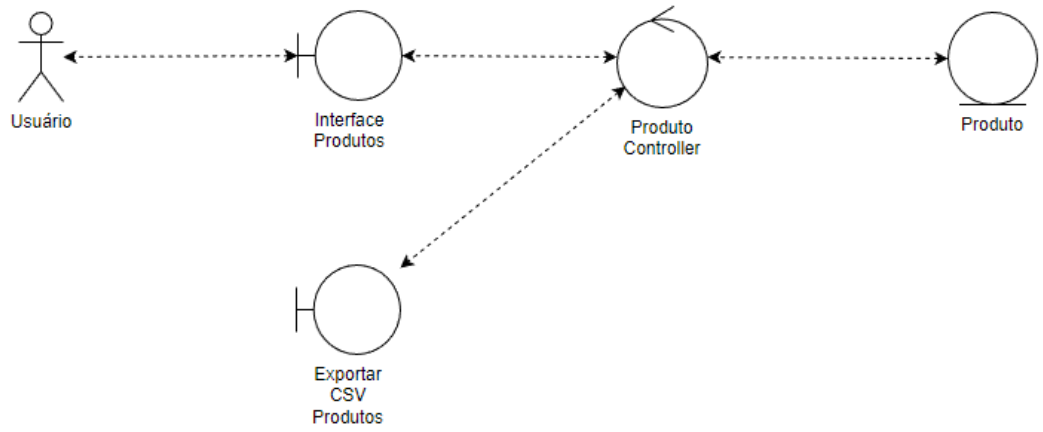


Fonte: O Autor (2023).

5.2.9.2 Diagrama de Robustez

Na Figura 31 é apresentado o diagrama de robustez do caso de uso 6 - Exportar planilha de produtos.

Figura 31 – Diagrama de Robustez 6 - Exportar planilha de produtos

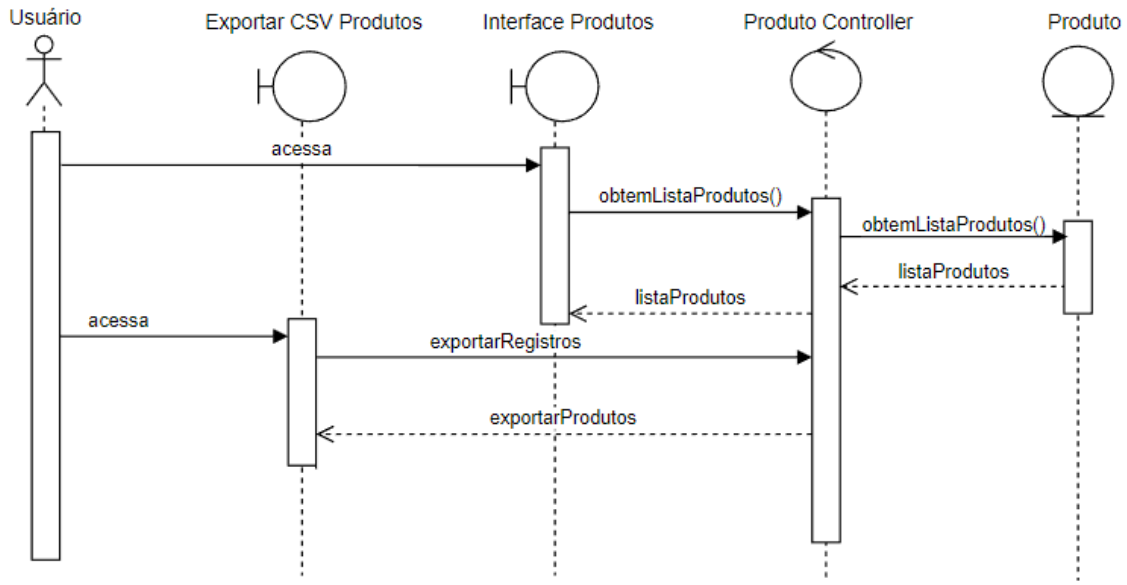


Fonte: O Autor (2023).

5.2.9.3 Diagrama de Sequência

Na Figura 32 é apresentado o diagrama de sequência do caso de uso 6 - Exportar planilha de produtos.

Figura 32 – Diagrama de Sequência 6 - Exportar planilha de produtos



Fonte: O Autor (2023).

5.2.9.4 Cenários

Cenário de sucesso:

- 1 - Usuário acessa o sistema.
- 2 - Usuário acessa a aba Produtos.
- 3 - Usuário clica em Exportar CSV.
- 4 - Sistema apresenta modal para confirmar a exportação.
- 5 - Usuário confirma a exportação e inicia o download da planilha.

Fluxo alternativo 1:

- 1 - Usuário acessa o sistema.
- 2 - Usuário acessa a aba Produtos.
- 3 - Usuário clica em Exportar CSV.
- 4 - Sistema apresenta mensagem informando que não existem produtos cadastrados.

5.2.10 Requisito R7 - Importar planilha de produtos

O usuário deve conseguir importar um arquivo em CSV contendo produtos e seus estoques: O usuário deve conseguir acessar o sistema, acessar a aba Produtos. Nesse local ele clica no botão Importar CSV. É apresentado um modal para o usuário anexar o CSV e confirmar a importação. Caso o arquivo importado contenha erros, é apresentado uma mensagem informando. Caso contrário, é apresentado uma mensagem de sucesso na importação.

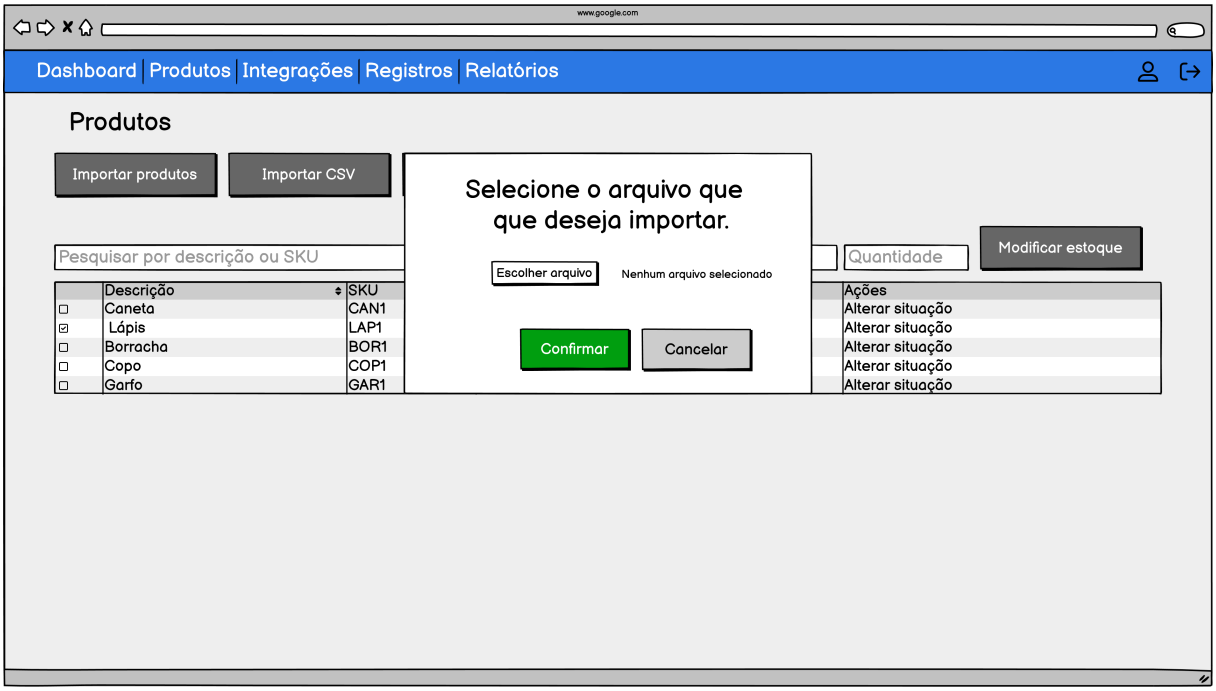
5.2.10.1 Protótipo da interface

Na Figura 33 é apresentado o protótipo de interface. A imagem apresenta o modal que será apresentado para confirmar a exportação do arquivo CSV contendo o histórico de alterações.

5.2.10.2 Diagrama de Robustez

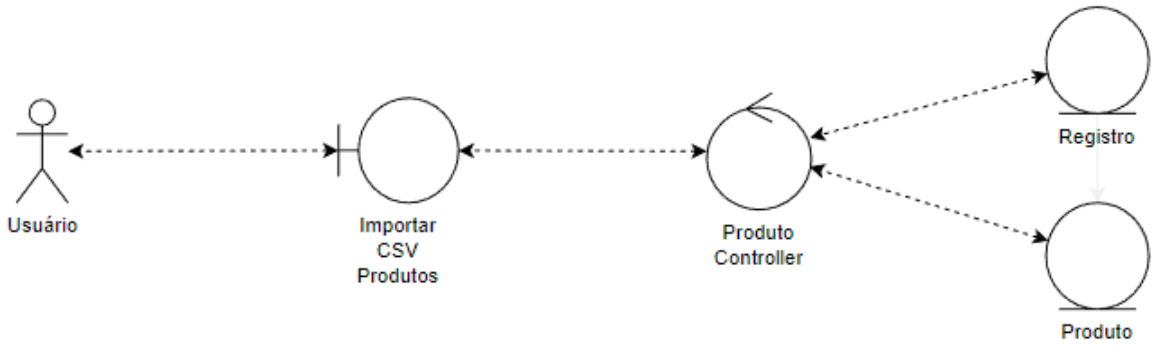
Na Figura 34 é apresentado o diagrama de robustez do caso de uso 7 - Importar planilha de produtos.

Figura 33 – Requisito R7 - Importar planilha de produtos



Fonte: O Autor (2023).

Figura 34 – Diagrama de Robustez 7 - Importar planilha de produtos



Fonte: O Autor (2023).

5.2.10.3 Diagrama de Sequência

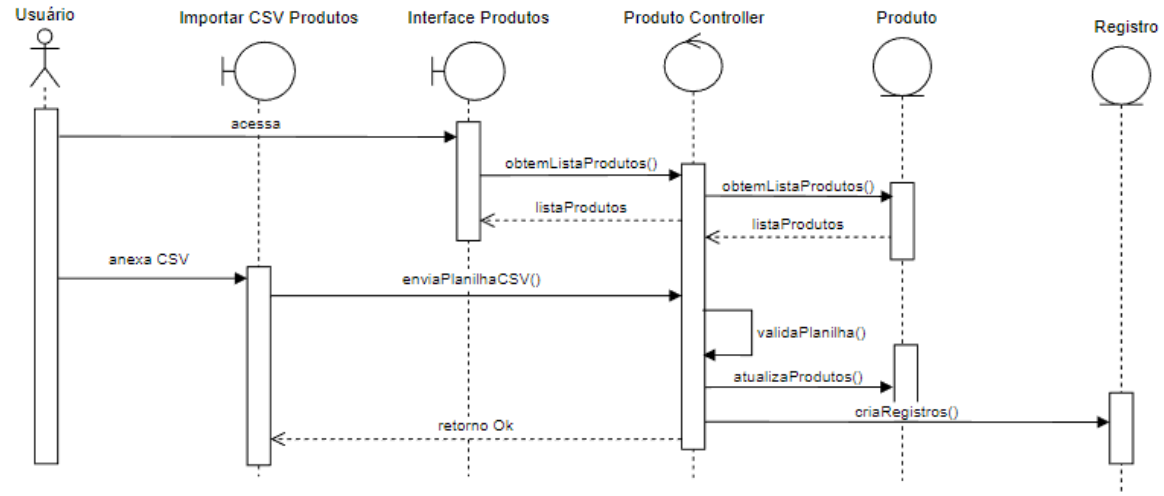
Na Figura 35 é apresentado o diagrama de sequência do caso de uso 7 - Importar planilha de produtos.

5.2.10.4 Cenários

Cenário de sucesso:

- 1 - Usuário acessa o sistema.

Figura 35 – Diagrama de Sequência 7 - Importar planilha de produtos



Fonte: O Autor (2023).

- 2 - Usuário acessa a aba Produtos.
- 3 - Usuário clica em Importar CSV.
- 4 - Sistema apresenta modal para anexar o arquivo e confirmar a exportação.
- 5 - Usuário anexa o arquivo e confirma a importação.
- 6 - Sistema valida o arquivo e retorna uma mensagem de sucesso.
- 7 - Produtos do sistema são atualizados conforme a planilha.

Fluxo alternativo 1:

- 1 - Usuário acessa o sistema.
- 2 - Usuário acessa a aba Produtos.
- 3 - Usuário clica em Importar CSV.
- 4 - Sistema apresenta modal para anexar o arquivo e confirmar a exportação.
- 5 - Usuário não anexa o arquivo e confirma a importação.
- 6 - Sistema retorna mensagem informando a necessidade de anexar o arquivo.

Fluxo alternativo 2:

- 1 - Usuário acessa o sistema.

- 2 - Usuário acessa a aba Produtos.
- 3 - Usuário clica em Importar CSV.
- 4 - Sistema apresenta modal para anexar o arquivo e confirmar a exportação.
- 5 - Usuário anexa o arquivo e confirma a importação.
- 6 - Sistema retorna mensagem informando que o arquivo contém erros.

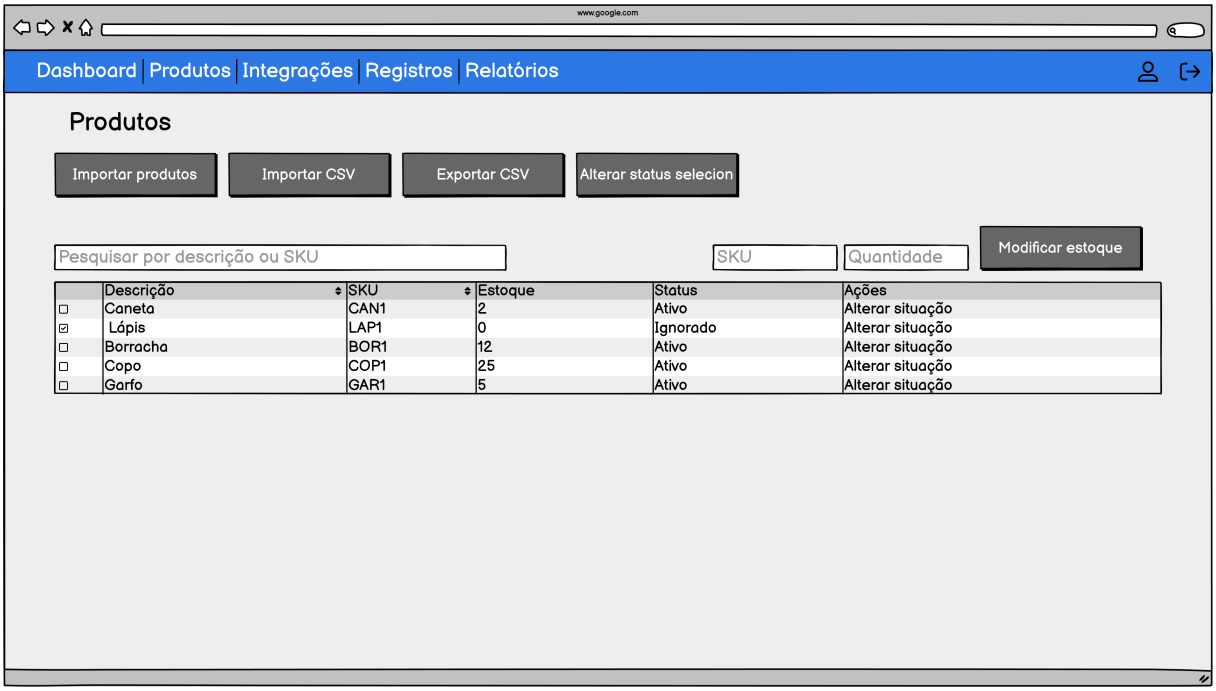
5.2.11 Requisito R8 - Alterar status de produtos

O usuário deve conseguir alterar o status de um produto no sistema para o sistema ignorá-lo e não atualizar o estoque no Bling: O usuário deve conseguir acessar o sistema, acessar a aba Produtos. Ali o usuário pode selecionar os produtos e clicar no botão "Alterar status selecionados". Dessa forma o status dos produtos selecionados é alterado para "Inativo", deixando assim de sincronizar estoques desses produtos de forma automática.

5.2.11.1 Protótipo da interface

Na Figura 36 é apresentado o protótipo de interface. A imagem apresenta o modal que será apresentado para confirmar a exportação do arquivo CSV contendo o histórico de alterações.

Figura 36 – Requisito R8 - Alterar status de produtos

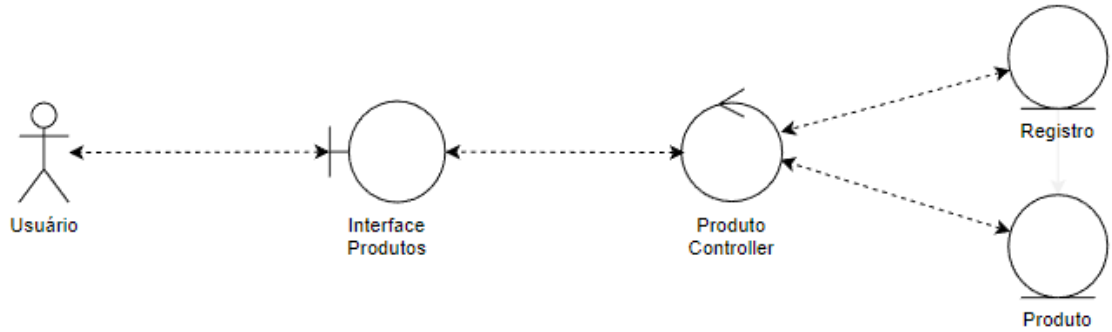


Fonte: O Autor (2023).

5.2.11.2 Diagrama de Robustez

Na Figura 37 é apresentado o diagrama de robustez do caso de uso 8 - Alterar status de produtos.

Figura 37 – Diagrama de Robustez 8 - Alterar status de produtos

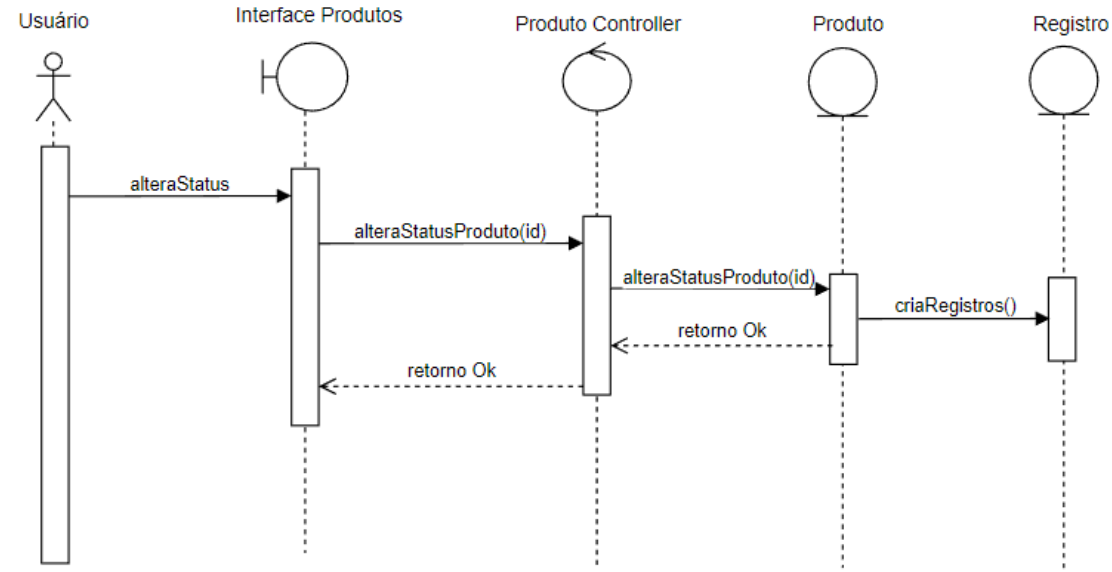


Fonte: O Autor (2023).

5.2.11.3 Diagrama de Sequência

Na Figura 38 é apresentado o diagrama de sequência do caso de uso 8 - Alterar status de produtos.

Figura 38 – Diagrama de Sequência 8 - Alterar status de produtos



Fonte: O Autor (2023).

5.2.11.4 Cenários

Cenário de sucesso:

- 1 - Usuário acessa o sistema.
- 2 - Usuário acessa a aba Produtos.
- 3 - Usuário seleciona alguns produtos e clica no botão "Alterar status selecionados".
- 4 - Sistema altera o status dos produtos selecionados e apresenta mensagem de sucesso.

5.2.12 Requisito R9 - Alterar estoque de um produto

O usuário deve conseguir alterar o estoque de um produto através do sistema para atualizar as contas do Bling: O usuário deve conseguir acessar o sistema, acessar a aba Produtos. Ali ele pode informar o SKU do produto que deseja alterar o estoque e a quantia que deseja inserir no produto. Após, ao clicar em "Modificar estoque"o sistema mostra um modal para confirmar a alteração. Ao confirmar a operação o estoque do produto é atualizado nas demais contas integradas.

5.2.12.1 Protótipo da interface

Na Figura 39 é apresentado o protótipo de interface. A imagem apresenta o modal que será apresentado para confirmar a alteração de estoque. Nesse modal apresenta o estoque atual e o estoque que será atualizado no produto.

5.2.12.2 Diagrama de Robustez

Na Figura 40 é apresentado o diagrama de robustez do caso de uso 9 - Alterar estoque de um produto.

5.2.12.3 Diagrama de Sequência

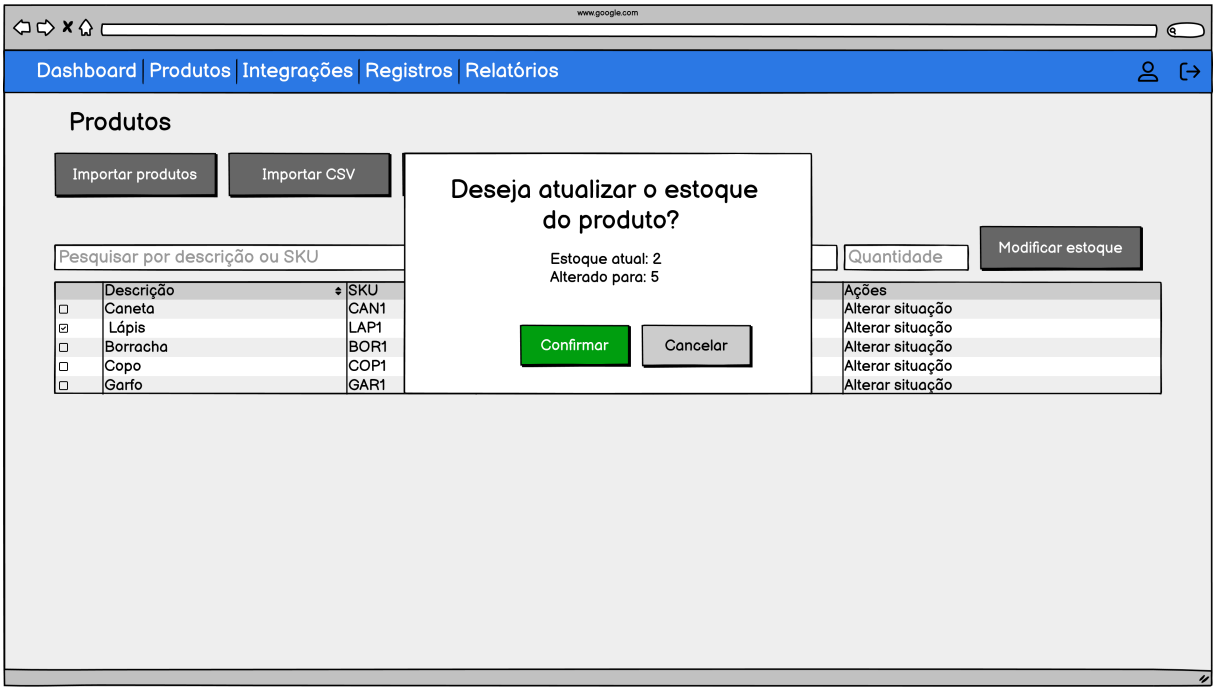
Na Figura 41 é apresentado o diagrama de sequência do caso de uso 9 - Alterar estoque de um produto.

5.2.12.4 Cenários

Cenário de sucesso:

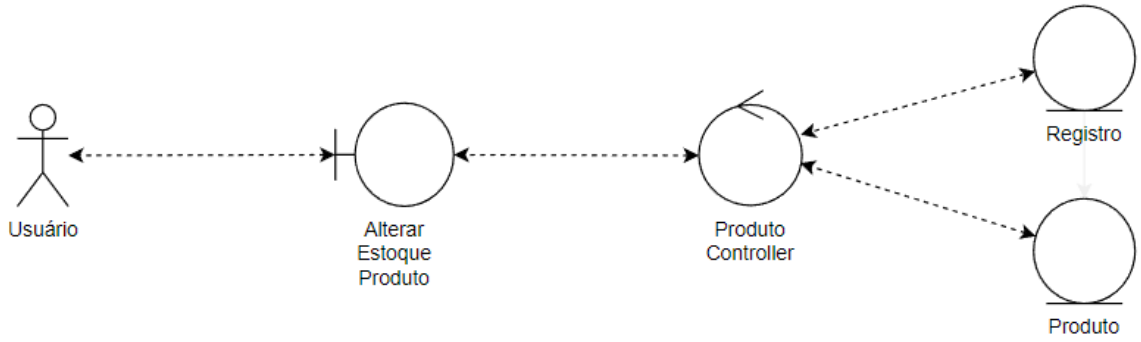
- 1 - Usuário acessa o sistema.
- 2 - Usuário acessa a aba Produtos.

Figura 39 – Requisito R9 - Alterar estoque de um produto



Fonte: O Autor (2023).

Figura 40 – Diagrama de Robustez 9 - Alterar estoque de um produto



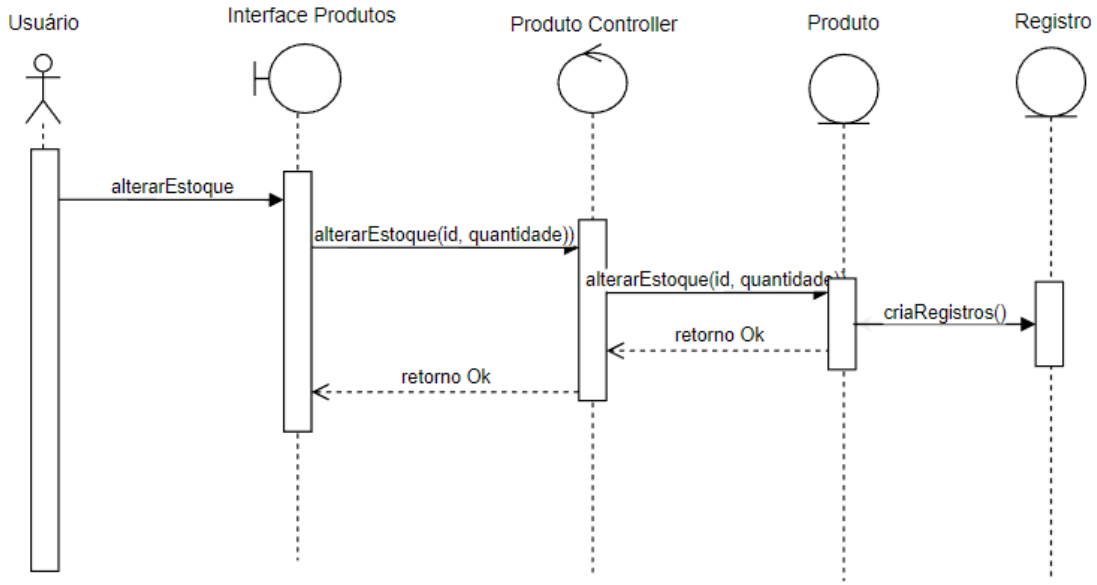
Fonte: O Autor (2023).

- 3 - Ususário preenche o SKU do produto e a quantidade corretamente.
- 4 - Usuário clica no botão "Modificar Estoque".
- 5 - Sistema apresenta modal para confirmar a alteração.
- 6 - Usuário confirma a alteração e o produto tem o estoque atualizado.

Fluxo alternativo 1:

- 1 - Usuário acessa o sistema.

Figura 41 – Diagrama de Sequência 9 - Alterar status de produtos



Fonte: O Autor (2023).

- 2 - Usuário acessa a aba Produtos.
- 3 - Ususário preenche o SKU do produto e a quantidade de forma incorreta.
- 4 - Usuário clica no botão "Modificar Estoque".
- 5 - Sistema apresenta modal para confirmar a alteração.
- 6 - Usuário confirma a alteração e o sistema apresenta mensagem informando que os dados foram preenchidos incorretamente.

6 DESENVOLVIMENTO DO PROTÓTIPO

Foi realizado o desenvolvimento do protótipo com base nos casos de uso, requisitos, protótipos de interface e artefatos Iconix planejados e criados anteriormente. Durante essa etapa ocorreu o desenvolvimento das interfaces de usuário, criação de testes unitários e desenvolvimento das funcionalidades do sistema. Também durante essa etapa ocorreram diversos refinamentos da proposta inicial.

6.1 PREPARAÇÃO DO AMBIENTE

O primeiro passo para iniciar a criação do protótipo foi preparar o ambiente local. O *framework* Laravel forneceu diversas facilidades nesse sentido. Optou-se pela configuração do ambiente utilizando o Laravel Breeze[1], juntamente com o kit de React/Vue.

O Laravel Breeze já fornece uma aplicação dispondo de sistema de autenticação e autorização de usuário, com rotas, *controllers* e *views* necessárias para registro e login. Dessa forma, a utilização dessa ferramenta permitiu economizar tempo no desenvolvimento do protótipo.

6.2 INTERFACES DE USUÁRIO

Optou-se por iniciar o desenvolvimento com a criação de todas as interfaces do sistema necessárias para a interação com o *middleware*, desde configuração até monitoramento. Isso auxiliou a amadurecer ainda mais os requisitos e casos de uso, localizar pontos de melhoria da própria interface e proporcionou a chance de prever com maior exatidão como seria o *backend* do protótipo.

Essa etapa iniciou com uma revisão dos protótipos de interface criados até o momento a fim de localizar elementos que tivessem reuso em diferentes locais. Dessa forma, esses elementos se transformaram em potenciais componentes de interface.

Componentes de interface do *framework* Vue podem ser brevemente descritos como blocos de código reutilizável. Ao reutilizar um componente aplicamos o princípio DRY (*Don't Repeat Yourself*, em português Não Se Repita), que promove a consistência, facilita a manutenção e acelera o desenvolvimento de software, uma vez que a alteração feita em um componente reflete a mudança em todas as instâncias onde esse componente é utilizado (THOMAS; HUNT, 2020).

Outra vantagem de usar componentes foi a possibilidade de atingir um nível maior de desacoplamento, permitindo que cada componente encapsule sua funcionalidade de forma in-

[1] https://laravel.com/docs/10.x/starter-kits#breeze-and-inertia

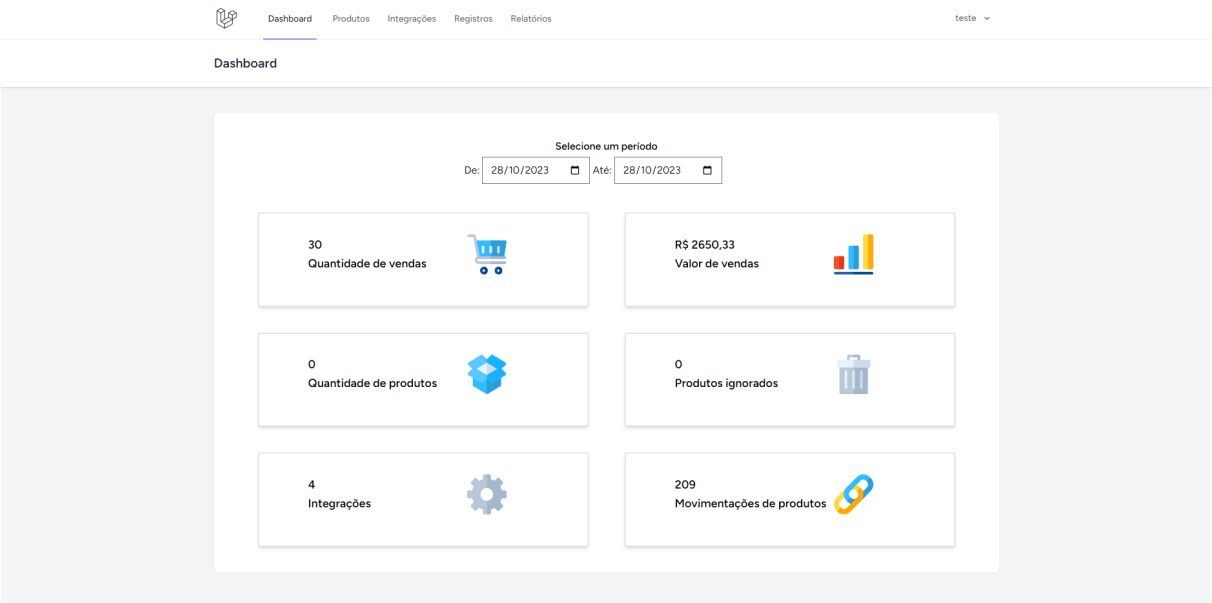
dependente. Isso aumentou a modularidade do sistema, tornando-o mais flexível e adaptável a mudanças.

No total foram utilizados 18 componentes básicos de interface. Esses componentes foram construídos utilizando Vue e são de variados tipos e com diferentes funções, como:

- Componentes de ação: Buttons e Links.
- Componentes de formulário: Checkbox, Dropdown, Input, InputLabel e DatePicker.
- Componentes de navegação: NavLink.
- Componentes informativos: Card, Modal.
- Componentes de listagem: Table.

Para a estilização dos componentes foi utilizada a biblioteca de CSS chamada *Tailwind*. Essa biblioteca, em conjunto com a componentização do Vue permitiu atingir maior coesão visual e comportamental da interface.

Figura 42 – Tela de dashboard



Fonte: O Autor (2023).

A Figura 42 é um exemplo do resultado final após o aperfeiçoamento das interfaces prototipadas. Essa é a tela de *dashboard* apresentada ao usuário assim que ele efetua o login no sistema. Nessa tela é possível visualizar três componentes sendo aplicados: O NavLink na parte superior listando abas com opções para acessar outros módulos do sistema, o DatePicker para filtrar pela data desejada e seis Card que apresentam o resumo dos dados.

Inicialmente as interfaces foram criadas sem obter dados reais do servidor. Todos os dados apresentados pela interface nesse estágio da prototipação eram gerados de forma estática ou simulados para fins de demonstração e teste.

6.3 TESTES

Após a etapa de construção das interfaces, foi iniciado o processo de escrita dos testes. A metodologia do TDD orienta que antes do desenvolvimento de funcionalidades, deve ser feita a codificação de testes unitários. Assim, foi separado um período específico no cronograma para esse processo.

O esforço para criar uma cobertura de testes foi aplicado especialmente em duas partes da estrutura do projeto Laravel: No diretório Http e no diretório *Services*. O diretório Http contém a camada de *Controllers*, que é responsável por lidar com as requisições HTTP. A cobertura de testes nessa área foi essencial para garantir que as funcionalidades da aplicação respondessem corretamente às solicitações que iria receber. Já o diretório *Services* contém as classes de serviço que desempenham as funcionalidades essenciais do protótipo e que são chamadas pela camada de *Controllers*.

O foco foi voltado para a criação de testes do tipo unitário, que testa apenas um método, normalmente validando sua entrada e saída. Foram codificados 45 testes, somando 76 verificações. A lista completa de testes está disponível para consulta no Anexo A. Cada verificação serve para validar se um determinado comportamento ou resposta está de acordo com o esperado. Cada teste precisa ter uma ou mais verificações. Um exemplo seria um método que efetua a soma de dois valores. Um teste para cobrir esse método poderia verificar se, passando dois valores o resultado retornado por esse método seria igual ao esperado.

O Algoritmo 2 representa um desses testes. Nesse trecho de código é realizado um teste para verificar se o usuário consegue realizar a autenticação do sistema utilizando a tela de login. No início desse teste é realizada a criação de um usuário temporário no banco de dados. Após, é executada uma requisição ao *endpoint* /login passando como parâmetros o email e senha desse usuário. Por fim, são feitas duas verificações. A primeira verifica se após efetuar essa requisição o usuário estará autenticado no sistema. E a segunda verifica se após a autenticação o usuário é redirecionado para a página inicial do sistema.

Algoritmo 2 – Teste unitário parar verificar autenticação

```
1
2 public function test_users_can_authenticate_using_the_login_screen(): void
3 {
4     $user = User::factory()->create();
5
6     $response = $this->post('/login', [
7         'email' => $user->email,
```

```
8           'password' => 'password',
9       ]);
10
11      $this->assertAuthenticated();
12      $response->assertRedirect(RouteServiceProvider::HOME);
13  }
```

Fonte: O Autor (2023)

No ambiente de desenvolvimento web, quando algum trecho de código que roda no lado do servidor possui algum erro, normalmente o servidor responderá com um código HTTP do tipo 500, que significa um erro interno no servidor. Um exemplo de teste simples que pode identificar esse tipo de erro é o do Algoritmo 3. Nesse teste é realizada uma requisição para a tela de *dashboard* e após é verificado se o código retornado foi o 200, indicando que a requisição foi tratada corretamente pelo servidor.

Algoritmo 3 – Teste unitário parar verificar renderização correta da interface do menu inicial

```
1
2   public function test_dashboard_screen_can_be_rendered(): void
3   {
4       $response = $this->get('/register');
5       $response->assertStatus(200);
6   }
```

Fonte: O Autor (2023)

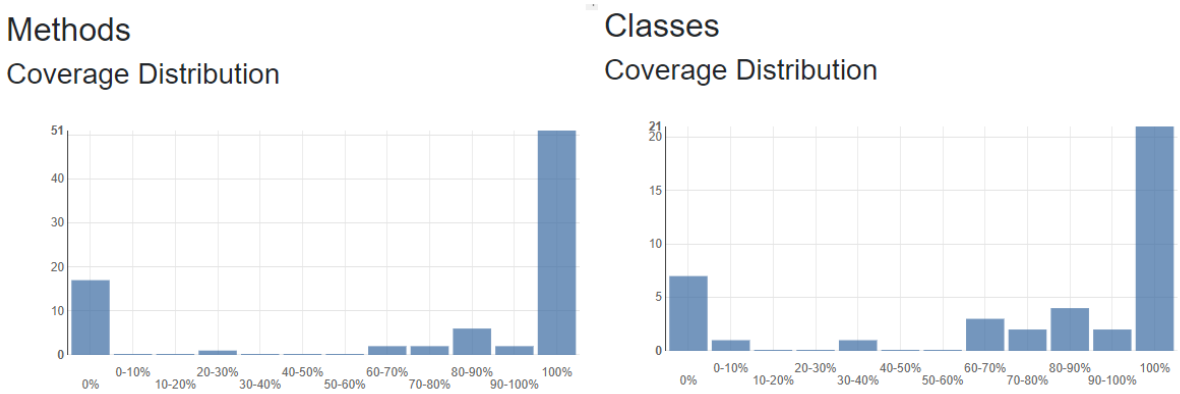
Esses testes foram desenvolvidos aplicando o princípio do Desenvolvimento Orientado a Testes (TDD). Esse processo se inicia pela criação de um novo teste seguido pela execução de todos os testes, criando dessa forma um teste que irá falhar. Posteriormente, são realizadas as implementações de código necessárias para fazer com que esse novo teste funcione. Por fim, são aplicadas refatorações nesse código, com o objetivo de torná-lo mais claro e otimizado.

Para facilitar o desenvolvimento e execução dos testes foi utilizada a ferramenta PHPUnit, juntamente com a extensão xDebug do PHP. Além de permitir a execução dos testes de forma simplificada, essa ferramenta permitiu a geração de alguns relatórios que apresentam a taxa de cobertura dos testes. Na Figura 43 e Figura 44 é possível verificar esses indicadores.

A Figura 43 demonstra em forma de gráfico a quantidade de métodos e classes e a porcentagem de linhas cobertas por testes de cada um deles.

Já a Figura 44 representa a quantidade de linhas cobertas por testes com base nos diretórios do projeto. Nessa figura é possível conferir que de um total de 403 linhas totais nos diretórios Http e Services, os testes cobriram 297, resultando numa métrica de 81,89% do código coberto por um ou mais testes. E de um total de 76 funções e métodos, 46 deles foram testados. Isso equivale a 60,53% das funções e métodos do protótipo.

Figura 43 – Cobertura de testes - Classes e métodos



Fonte: O Autor (2023).

Figura 44 – Cobertura de testes - Nível de cobertura

	Code Coverage					
	Lines			Functions and Methods		
Total		81.89%	297 / 403		60.53%	46 / 76
Http		80.80%	181 / 224		56.25%	27 / 48
Services		83.24%	149 / 179		67.86%	19 / 28

Legend

Low: 0% to 50% **Medium**: 50% to 90% **High**: 90% to 100%

Fonte: O Autor (2023).

Essa discrepância entre a quantidade de linhas de código e a quantidade de funções cobertas por testes ocorre devido a maior parte dos testes terem sido criados para cobrir métodos maiores em comparação com os não testados. Essa abordagem foi escolhida porque vários métodos foram considerados simples a ponto de não demandarem testes nesse primeiro momento, sendo reservado o tempo para métodos mais extensos ou críticos, relacionados com funcionalidades essenciais do sistema.

Apesar da prática de iniciar o desenvolvimento de funcionalidades com a criação dos testes ter sido aplicada, por vezes o próprio teste precisou ser revisado e ajustado devido a mudanças no entendimento da funcionalidade ao longo do processo de desenvolvimento. Essa flexibilidade e capacidade de adaptação demonstrou a importância de um ciclo ágil no desenvolvimento de software.

6.4 FLUXOS DE UTILIZAÇÃO

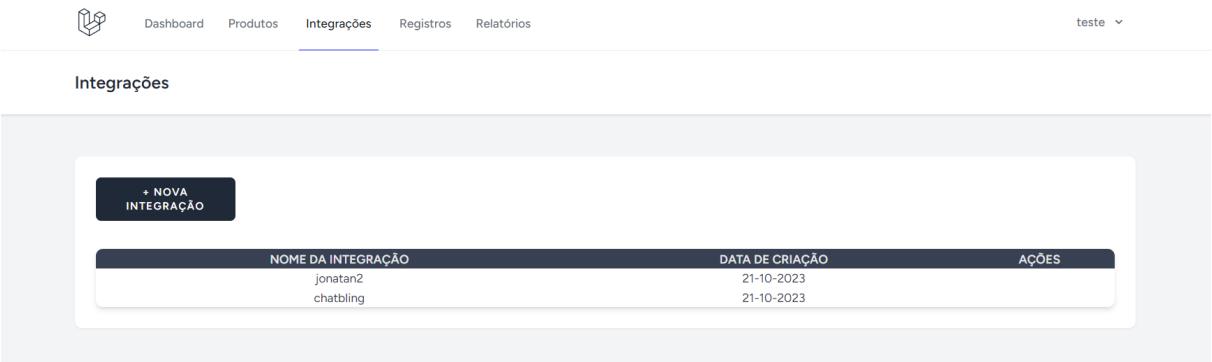
Após as etapas de criação das interfaces e implementação dos testes e das funcionalidades, foram executados os fluxos principais de utilização do sistema para testar de forma manual se as funcionalidades satisfazem os requisitos e estão de acordo com os casos de uso. Também serviu para documentar como o usuário pode utilizar o sistema. Os fluxos secundários

foram suprimidos e foram mantidos apenas os fluxos principais, julgados necessários. Foram considerados apenas os casos de sucesso, sem apresentar os fluxos alternativos.

6.4.1 Cadastro de integração

Após ter criado seu registro e acessar o sistema, a primeira etapa para o usuário utilizar funcionalidades do sistema é a criação de integração com uma conta Bling. Isso pode ser feito acessando a aba Integrações e clicando no botão "+ Nova Integração", conforme a Figura 45

Figura 45 – Tela de integrações



Fonte: O Autor (2023).

Ao clicar nesse botão surgirá um modal onde é possível preencher o nome da integração e a chave API que será utilizada nas chamadas. Após, clicando em salvar já estará criada a integração. Esse processo pode ser repetido para adicionar outras integrações com demais contas Bling que esse usuário possuir.

6.4.2 Configuração do webhook

Para a comunicação entre as contas Bling e o *middleware* funcionar também é preciso configurar o *webhook* da conta Bling. Para essa configuração, é preciso acessar uma página específica dentro do sistema Bling e inserir a URL do *middleware* que receberá as requisições sempre que um evento de alteração de estoque for disparado no Bling. Essa URL deve ser configurada no Bling seguinto esse formato: {domínio do middleware}/webhooks/{chave API da conta Bling}.

Dessa forma pode ser utilizada a própria chave API do sistema Bling para direcionar os eventos daquela conta no *middleware*.

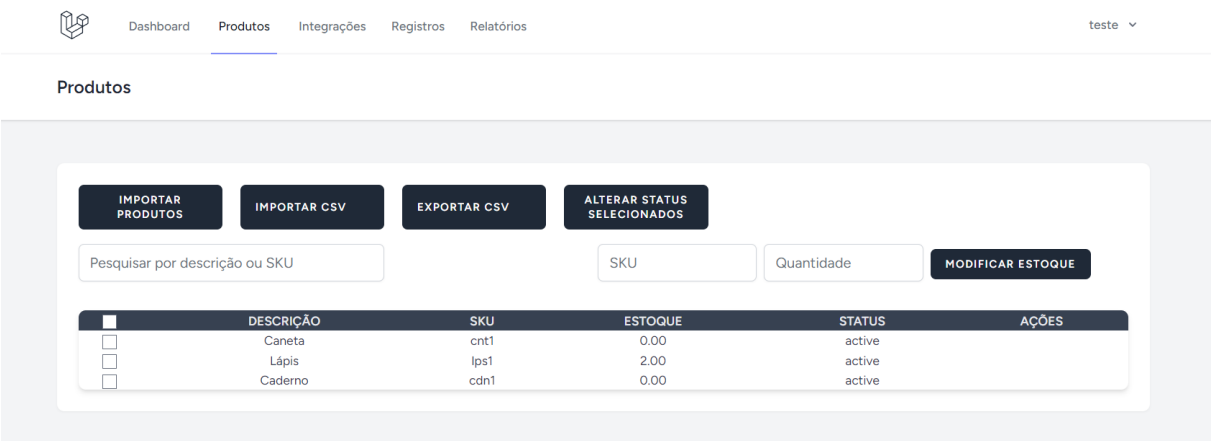
6.4.3 Importação dos produtos

Depois das integrações terem sido configuradas é possível realizar a importação de produtos de cada uma delas. A importação de produtos pode ser feita acessando a aba Produtos, conforme a Figura 46 e clicando no botão "Importar produtos". Surgirá um modal solicitando

de qual das contas integradas deseja importar os produtos. Ao selecionar a integração e iniciar a importação, o sistema realizará requisições ao sistema Bling e registrará esses produtos no baco de dados, com sua situação e estoque, conforme estiverem cadastrados no Bling.

Esse procedimento deve ser realizado uma vez para cada conta do Bling que desejar importar os produtos. Durante esse processo, produtos que tiverem o código SKU igual terão seus dados de situação e estoque sobrescritos pelos da importação. Produtos com código SKU diferente dos já cadastrados no *middleware* serão cadastrados como novos produtos.

Figura 46 – Tela de produtos



Fonte: O Autor (2023).

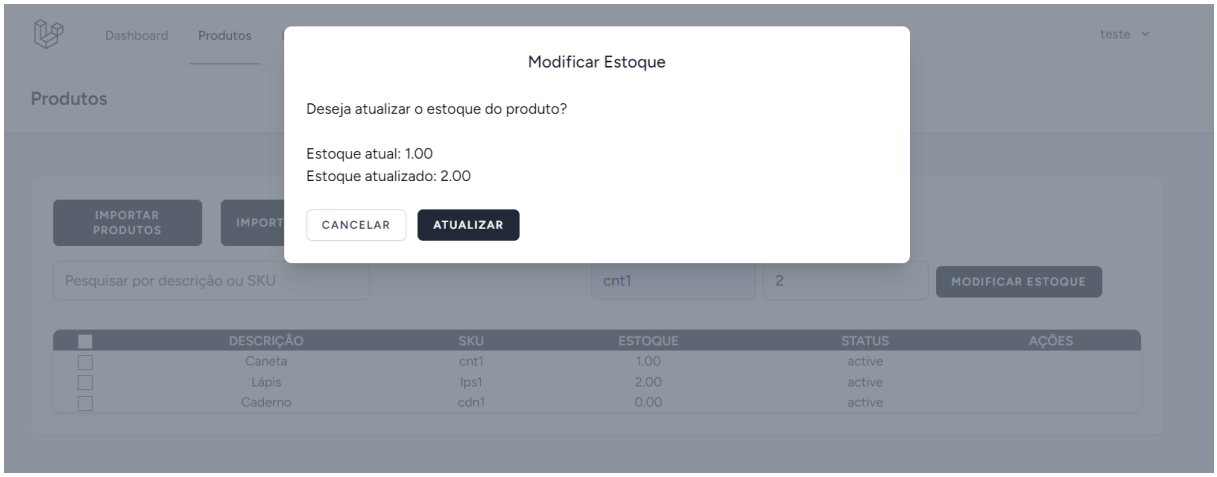
6.4.4 Alteração automática de estoque

Nesse ponto o *middleware* já está configurado e a automação do estoque já está ativa. Os produtos que tiverem sido importados e estiverem com a situação ativa já terão o estoque atualizado sempre que tiverem alteração em uma das contas do Bling. Isso se dará por meio do recebimento do evento através do *webhook* e reenvio dessa informação pelo *middleware* para as demais integrações.

6.4.5 Alteração manual de estoque

Além da alteração automática que ocorre caso uma das contas do Bling tenha uma alteração de estoque, é possível fazer ajustes manuais de estoque diretamente pelo sistema. Isso pode ser feito inserindo o código SKU do produto e informando a quantidade desejada de estoque. Após, clicando no botão "Modificar estoque"surgirá um modal, conforme a Figura 47, mostrando a quantidade antiga de estoque desse produto e a quantidade que está sendo atualizada. Ao clicar no botão "Atualizar"o sistema enviará requisições para atualizar esse estoque em todas as integrações.

Figura 47 – Modal de confirmação de alteração manual de estoque



Fonte: O Autor (2023).

6.4.6 Alteração em massa de estoque

Também é possível realizar alterações em massa no estoque dos produtos. Essa alteração pode ser feita ao exportar a planilha modelo no formato CSV, realizar as alterações de estoque necessárias e importá-la novamente no sistema. Essas ações podem ser realizadas usando os botões "Exportar CSV"e "Importar CSV"da Figura 46.

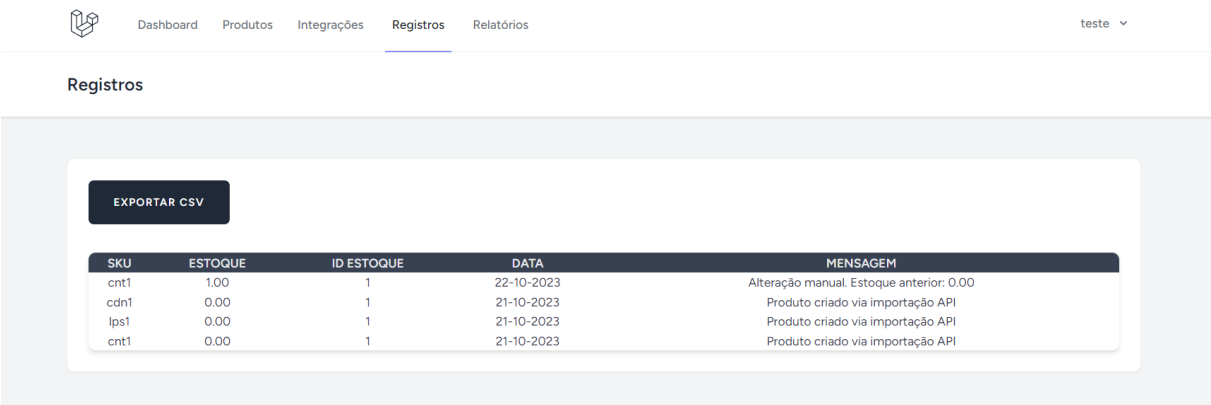
6.4.7 Alteração status dos produtos

Caso o usuário possua algum produto específico que deseja que o sistema ignore alterações de estoque é possível desativar esse produto. Para isso, pode selecionar a caixa de seleção ao lado do produto e após clicar em "Alterar status selecionados", conforme a Figura 46. Dessa forma o sistema atualizará a situação desse produto para inativo e deixará de atualizar seu estoque.

6.4.8 Consulta de registros

Para fornecer maior transparência das alterações que o sistema realiza nos estoques de outros sistemas é possível verificar os registros gerados. Para isso é possível acessar a aba Registro. Ali serão listadas alterações de estoque mostrando o nome e código e valor atualizado destoque, juntamente com a data e uma mensagem da origem da alteração. Esses registros estarão vinculados ao usuário que efetuou as movimentações e só são apresentados para ele ao acessar essa página, com cada usuário tendo seus próprios registros. A imagem Figura 48 representa essa tela e as opções comentadas acima.

Figura 48 – Tela de registros



Fonte: O Autor (2023).

7 CONSIDERAÇÕES FINAIS

Esse trabalho visava resolver um problema relacionado ao tema de integração de dados entre sistemas. O problema em questão era relacionado a falta de integração de estoques do sistema ERP chamado Bling. Para fundamentar conceitos importantes desse tema foi necessário iniciar a busca de conhecimentos mais aprofundados sobre métodos de comunicações entre sistemas, arquiteturas de sistemas e um breve resumo sobre ERPs.

Depois de efetuados estes estudos, foi iniciada a etapa de análise e levantamento de requisitos e a criação de casos de uso. Com essas informações foi possível perceber que um middleware poderia ser construído para trabalhar em segundo plano e propagar informações de estoque entre as diversas contas desse ERP. Esse processo necessitaria a comunicação do protótipo com esse sistema através de requisições HTTP usando o padrão REST.

Além do integração e comunicação das informações de estoque de forma passiva, também surgiu a possibilidade de criar algumas funcionalidades adicionais que poderiam trazer maior utilidade ao protótipo e auxiliaria o usuário. Algumas dessas funcionalidades foram a disponibilidade de alterações do estoque iniciadas diretamente através da interface do middleware, ao invés de apenas propagar as alterações que ocorressem e configurações para ignorar a propagação de estoque de determinados produtos. Também se tornou possível modificar o estoque dos produtos em massa via importação e exportação de planilhas e a consulta de registros de alteração, permitindo ao usuário visualizar alterações juntamente com a origem dela.

Para implantar o protótipo gerado a partir desse trabalho e passar a utilizá-lo de forma comercial seria preciso aprimorar as funcionalidades disponibilizadas até o momento, para tornar a experiência do usuário mais agradável. Caso o ERP Bling decidisse utilizar essa ferramenta, também seria preciso reavaliar as interfaces para aderirem ao design do próprio ERP. Seria necessário investir tempo e recursos para revisar questões de segurança de dados sensíveis e testar a ferramenta com níveis mais altos de tráfego de dados para validar sua escalabilidade.

7.1 DESAFIOS ENCONTRADOS

Um dos principais motivos para escolha das tecnologias utilizadas no desenvolvimento desse trabalho foi me aprofundar mais em *frameworks* como Laravel e Vue e aplicar metodologias bastante desejáveis no mercado atual, como o TDD. Porém a falta de experiência nessas ferramentas dificultou um pouco a implementação de algumas funcionalidades. Por exemplo, a não utilização de todo potencial dessas ferramentas pode ter impactado na qualidade da solução, visto que algumas funcionalidades possivelmente não foram utilizadas em seu máximo, resultando em uma subutilização de recursos e, potencialmente, em oportunidades perdidas para aprimorar a eficácia do protótipo.

Além disso, a aplicação do TDD foi uma tarefa árdua, pois exigiu uma mudança significativa na forma como desenvolvemos software. Adotando o TDD foi necessário pensar cuidadosamente nos requisitos antes da implementação. Foi necessária uma transição da abordagem tradicional de desenvolvimento para uma mentalidade orientada aos testes. Isso gerou uma curva de aprendizado mais acentuada e alguns retrabalhos adicionais durante o processo de desenvolvimento.

7.2 CONCLUSÃO

Durante a etapa de levantamento dos requisitos e casos de uso, a utilização de artefatos UML da metodologia ICONIX proveram uma base sólida para a tarefa de abstrair e modelar o sistema. Esses artefatos forneceram uma estrutura visual, que garantiu que os requisitos fossem compreendidos, documentados e implementados de forma correta e eficaz. Por sua vez, a compreensão e documentação dos requisitos também foi vital para a construção da cobertura de testes necessária para a aplicação da metodologia TDD (Desenvolvimento Orientado a Testes), visto que torna-se ainda mais eficaz quando baseada em requisitos bem definidos e casos de uso sólidos, ambos apoiados pelos artefatos UML da metodologia ICONIX.

Em resumo, a combinação do uso de artefatos UML da metodologia ICONIX na fase de levantamento de requisitos, juntamente com a aplicação da metodologia TDD, permitiu uma abordagem mais sólida e eficaz para o desenvolvimento do protótipo e a busca por uma solução computacional adequada.

A criação dos protótipos de interface na fase inicial do projeto também foi interessante, pois auxiliou o amadurecimento dos casos de uso, facilitando o planejamento do fluxo do usuário dentro do sistema. Dessa forma foi possível obter uma experiência mais agradável para o usuário e validar os conceitos imaginados para as interfaces.

A utilização de tecnologias difundidas no mercado e com grandes comunidades também foi uma opção interessantes por uma série de fatores. Existe uma vasta quantidade de documentação sobre essas tecnologias disponível. A comunidade dessas tecnologias é de forma geral ativa e engajada, auxiliando na resolução de problemas específicos que possam ocorrer durante a utilização dessas ferramentas.

Ao final do trabalho é possível visualizar a solução completa, com todos objetivos principais atingidos e casos de uso implementados e funcionando conforme o planejado. As metodologias aplicadas auxiliaram na conclusão bem sucedida do trabalho, desde o planejamento da proposta até a etapa de validação da proposta por meio do desenvolvimento do protótipo. Os conhecimentos adquiridos ao longo do curso de Ciência da Computação mostraram-se fundamentais para compreender e resolver esse problema, permitir a realização desse projeto. Os conceitos obtidos através dos estudos e trabalhos forneceram base teórica e prática necessária para abordar esse desafios. Essa base de conhecimento se tornou um ativo valioso ao realizar

projetos e resolver problemas complexos como esse.

7.3 TRABALHOS FUTUROS

Como sugestões de melhoria e continuidade deste trabalho sugere-se:

- Comparação dos resultados obtidos ao criar uma solução semelhante utilizando outra arquitetura e ferramentas.
- Uma solução que envolva aplicar testes de integração e de interface além dos testes unitários.
- Aprimorar componentes de interface para fornecer maior variação visual e de comportamentos para eles.
- Fornecer integração de estoque com outros sistemas além do Bling.
- Aprimorar a reatividades das interfaces para diferentes tipos de telas.
- Disponibilizar relatórios em formato CSV e de gráficos para auxiliar o controle de movimentações ocorridas.

REFERÊNCIAS

CORREA, H. L.; CAON, M.; GIANESI, I. G. N. **Planejamento, Programação e Controle da Produção**: Mrp ii/erp - conceitos, uso e implantação, base para sap, oracle applications e outros softwares integrados de gestão. Sebastopol: Atlas, 2007.

DOGLIO, F. **REST API Development with Node.js: Manage and Understand the Full Capabilities of Successful REST Development**. 2nd ed. ed. [S.l.]: Apress, 2018.

ERINC, Y. K. **The Benefits of Going RESTful - What is REST and Why You Should Learn About It**: What is rest and why you should learn about it. [s.n.], 2020. Disponível em: <https://www.freecodecamp.org/news/benefits-of-rest/>. Acesso em: 11 jun. 2023.

ERL, T. **Web Services**: Uma introdução às tecnologias web services: Soa, soap, restful, wsdl e uddi. 2009. Disponível em: <https://www.devmedia.com.br/web-services/2873>.

FACEBOOK. **Launching Reels on Facebook in the US**. Menlo Park, 2021. Disponível em: <https://about.fb.com/news/2021/09/launching-reels-on-facebook-us/>. Acesso em: 06 abr. 2023.

FOWLER, M. **Richardson Maturity Model**. [s.n.], 2010. Disponível em: <martinfowler.com/articles/richardsonMaturityModel.html>. Acesso em: 11 jun. 2023.

____. **Integration Database**. Boston: [s.n.], 2015. Disponível em: <https://martinfowler.com/bliki/IntegrationDatabase.html>. Acesso em: 25 abr. 2023.

FREEMAN, S.; PRYCE, N. **Growing Object-Oriented Software, Guided by Tests**. [S.l.]: Addison-Wesley Professional, 2010.

GAMMA, E. *et al.* **Design Patterns**: elements of reusable object-oriented software. Westford: Addison-Wesley Professional, 2009.

GARBAR, D. **The Top 10 Advantages Of Using Laravel PHP Framework**. [s.n.], 2020. Disponível em: <https://belitsoft.com/blog/10-benefits-using-laravel-php-framework>. Acesso em: 28 maio 2023.

GARCIA, C. M.; ABILIO, R. **Integração entre Sistemas utilizando Web Services REST e SOAP: Um Relato Prático**. [s.n.], 2017. Disponível em: <https://www.fsma.edu.br/si/19edicao.html>. Acesso em: 11 jun. 2023.

GAZIS, A.; KATSIRI, E. **Middleware 101**: What to know now and for the future. 2022. Disponível em: <https://queue.acm.org/detail.cfm?id=3526211&doi=10.1145%2F3526211>.

HARRIS, C. **Microservices vs monolithic architecture**. [s.n.], 2022. Disponível em: <https://www.atlassian.com/microservices/microservices-architecture/microservicesvs-monolith>. Acesso em: 12 jun. 2023.

HOHPE, G.; WOOLF, B. **Enterprise Integration Patterns**: Designing, building, and deploying messaging solutions. Boston: Addison-Wesley Professional, 2003.

IBM. **RPC Model**. 2023. Disponível em: <https://www.ibm.com/docs/en/aix/7.1?topic=call-rpc-model>. Acesso em: 09 jun. 2023.

INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATíSTICA. **Internet já é acessível em 90,0% dos domicílios do país em 2021**. Rio de Janeiro, 2022.

JúNIOR, C. C. **Sistemas integrados de gestão ERP: uma abordagem gerencial**. 4. ed. [S.l.]: Editora Ibpex, 2015.

KARANAM, R. **Richardson Maturity Model**. [s.n.], 2019. Disponível em: <https://dzone.com/articles/rest-api-what-is-hateoas>. Acesso em: 11 jun. 2023.

LAAZIRIA, M. *et al.* **A Comparative study of PHP frameworks performance**. [s.n.], 2019. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2351978919303312>. Acesso em: 30 maio 2023.

LI, S. **A survey on security analysis of OAuth 2.0 framework**. 2019. Disponível em: <https://www.cs.purdue.edu/homes/li3944/blog/survey-security-analysis-Shuai.pdf>. Acesso em: 19 jun 2023.

LINTHICUM, D. S. **Enterprise Application Integration**. [S.l.]: Addison-Wesley Professional, 2000.

MOTA, A. **O que é mensageria e o que eu ganho com isso?** [s.n.], 2022. Disponível em: <https://br.sensedia.com/post/what-is-messaging>. Acesso em: 09 jun. 2023.

PATIL, D.; MASON, H. **Data Driven**: Creating a data culture. Sebastopol: O'Reilly Media, 2015.

PRESSMAN, R. S.; MAXIM, B. **Software Engine**: A practitioner's approach. 9. ed. [S.l.]: McGraw-Hill, 2019.

RAMOS, N. K.; YAMAGUCHI, C. K.; COSTA, U. M. da. **Tecnologia da Informação e Gestão do Conhecimento: Estratégia de competitividade nas organizações**. 2020. Disponível em: <https://ojs.brazilianjournals.com.br/ojs/index.php/BRJD/article/view/5830/5238>. Acesso em: 16 ago 2023.

RICE, C.; SIEBEL, T. M. **Digital transformation: survive and thrive in an era of mass extinction**. [S.l.]: RosettaBooks, 2019.

RICHARDS, M. **Software Architecture Patterns**: Understanding common architecture patterns and when to use them. [S.l.]: O'Reilly Media, 2015.

ROSENBERG, D.; COLLINS-COPE, M.; STEPHENS, M. **Agile Development with ICONIX Process**: People, process, and pragmatism. 1. ed. [S.l.]: Apress, 2005.

ROYCE, W. **Managing The Development Of Large Software Systems**. [S.l.]: TRW, 1970.

SCEARCE, T. **Pros and Cons of File Transfer in Enterprise Application Integration**. [s.n.], 2014. Disponível em: <https://blog.microfocus.com/pros-and-cons-of-file-transfer-in-enterprise-applicationintegration>. Acesso em: 26 maio 2023.

SOMMERVILE, I. **Engenharia de Software**. 9. ed. [S.l.]: Pearson Education, 2011.

STEC, A. **Database Design in a Microservices Architecture**. [s.n.], 2023. Disponível em: <https://www.baeldung.com/cs/microservices-db-design>. Acesso em: 25 maio 2023.

THOMAS, D.; HUNT, A. **The Pragmatic Programmer - 20th Anniversary Edition**. 2. ed. [S.l.]: Pearson, 2020.

TRIPATHY, A.; NAIK, K. **Software Evolution and Maintence**: A practitioner's approach. [S.l.]: John Wiley Sons, Inc., 2015.

TURBAN, E.; VOLONINO, L. **Tecnologia da Informação para Gestão**: Em busca de um melhor desempenho estratégico e operacional. Porto Alegre: Bookman Editora, 2013.

W3. **Simple Object Access Protocol (SOAP) 1.1**. 2000. Disponível em: <https://www.w3.org/TR/2000/NOTE-SOAP-20000508>. Acesso em: 10 jun. 2023.

____. **XML Schema Part 1: Structures Second Edition**. 2004. Disponível em: <https://www.w3.org/TR/2004/REC-xmlschema-1-20041028/>. Acesso em: 10 jun. 2023.

XU, L. D. **Enterprise Integration and Information Architecture: A Systems Perspective on Industrial Information Integration**. 1. ed. [S.l.]: Auerbach Publications,CRC Press, 2014. (Advances in Systems Science and Engineering (ASSE)).

ANEXO A – LISTA DE TESTES

Tabela 1 – Lista de testes criados para a cobertura de testes do protótipo.

(continua)

Número	Nome	Verificações
1	test_new_users_can_register	1
2	test_users_can_authenticate_using_the_login_screen	2
3	test_users_can_not_authenticate_with_invalid_password	1
4	test_email_verification_screen_can_be_rendered	1
5	test_email_can_be_verified	2
6	test_email_is_not_verified_with_invalid_hash	1
7	test_confirm_password_screen_can_be_rendered	1
8	test_password_can_be_confirmed	2
9	test_password_is_not_confirmed_with_invalid_password	1
10	test_reset_password_link_screen_can_be_rendered	1
11	test_reset_password_link_can_be_requested	1
12	test_reset_password_screen_can_be_rendered	1
13	test_password_can_be_reset_with_valid_token	3
14	test_password_can_be_updated	2
15	test_correct_password_must_be_provided_to_update_password	1
16	test_registration_screen_can_be_rendered	2
17	test_new_users_can_register	1
18	test_dashboard_screen_can_be_rendered	1
19	test_dashboard_controller_response_to_index	1
20	test_dashboard_filtered_with_invalid_date	5
21	test_dashboard_filtered_with_valid_date	1
22	test_integration_screen_can_be_rendered	3
23	test_create_new_integration	1
24	test_obtain_all_integrations_list	3
25	test_create_and_delete_new_integration	1
26	test_product_screen_can_be_rendered	2
27	test_product_controller_response_to_index	3
28	test_get_product_by_name_or_sku	3
29	test_show_response_product_controller	1
30	test_can_export_csv_of_products	1

Tabela 1 – Lista de testes criados para a cobertura de testes do protótipo.

(conclusão)

Número	Nome	Verificações
31	test_can_create_new_product	1
32	test_get_product_by_name	1
33	test_get_product_by_sku	1
34	test_get_all_products_from_api	2
35	test_save_products_from_api	1
36	test_health_of_api	1
37	test_generate_xml_to_update_stock_from_api	2
38	test_automatic_stock_sincronization	3
39	test_can_change_status_of_product	1
40	test_csv_of_product_exist	1
41	test_can_export_contend_of_csv_of_products	2
42	test_can_import_csv_of_products	4
43	test_can_change_product_stock_by_sku	1
44	test_obtain_all_products_list	2
45	test_compare_arrays_method_return	3

Fonte: O Autor (2023).