UNIVERSIDADE FEEVALE

RENAN HALBERT CARDOSO POSPICHIL

CONVERSÃO DE SISTEMAS WINFORMS PARA ASP.NET CORE

Novo Hamburgo

2021

RENAN HALBERT CARDOSO POSPICHIL

CONVERSÃO DE SISTEMAS WINFORMS PARA ASP.NET CORE

Trabalho de Conclusão de Curso, apresentado como requisito parcial à obtenção do grau de Bacharel em Ciência da Computação pela Universidade Feevale

Orientador: Dr. Gabriel Da Silva Simões

Novo Hamburgo

2021

AGRADECIMENTOS

Gostaria de agradecer a todos os que, de alguma maneira, contribuíram para a realização desse trabalho de conclusão, em especial:

Aos meus pais, Andreia e Reni, por sempre priorizarem meus estudos e incentivarem minha dedicação.

À minha esposa Marcia e à minha filha Mariana por me apoiarem no desenvolvimento deste trabalho.

Ao professor Gabriel da Silva Simões, pela sua disposição, paciência e por todas as sugestões concedidas durante a construção deste trabalho.

À empresa Sesiom por ter cedido o projeto no qual este trabalho utilizou como base.

Aos colegas de empresa, que contribuíram na parte de experimentos e respostas ao questionário.

A todos que, de alguma forma, contribuíram para a conclusão deste trabalho.

Muito obrigado!

RESUMO

Quando fala-se em sistemas computacionais, de modo geral, pensa-se em algo complexo e de imensa importância para as empresas. Na falta destes sistemas, muitas empresas teriam de suspender suas atividades já que, em geral, grande parte da gestão dos estoques e dos processos produtivos são mantidos por estes sistemas. Substituir um sistema pode ser um problema gigantesco já que, além do esforço em se instalar um novo sistema, também é necessário configurá-lo por completo, importando todo o histórico anterior, além de treinar os usuários. Este trabalho prototipou um *framework* capaz de transformar um sistema WinForms em um sistema Web de maneira que, além de manter a compatibilidade com dispositivos contemporâneos, como dispositivos móveis, possa também explorar recursos de servidores em nuvem. Um sistema legado convertido pelo protótipo desenvolvido foi avaliado por usuários que compararam diferentes pontos do sistema original com o sistema convertido, apresentando resultados promissores.

Palavras-chave: Nuvem. Conversão de sistemas. Sistema Web. WinForms. AspNet Core.

ABSTRACT

When talking about computer systems, in general, we think of something complex and of immense importance for companies. In the absence of these systems, many companies offer to suspend their activities since, in general, much of the inventory management and production processes are collected by these systems. Replacing a system can be a huge problem since, in addition to the effort to install a new system, it is also necessary to configure it completely, importing all previous history, in addition to training users. This work presents the development of a prototyped framework capable of transforming a WinForms system into a Web system so that, in addition to maintaining compatibility with contemporary devices, such as mobile devices, it can also exploit cloud server resources. A legacy system converted by the developed prototype was evaluated by users who compared different elements of the original system with the converted system, which presents promising results.

Keywords: Cloud. Conversion of systems. Web system. WinForms. AspNet Core.

LISTA DE FIGURAS

LISTA DE ABREVIAÇÕES E SIGLAS

API	*Application Programming Interfaces*
CPU	*Central Process Unit*
CSP	*Cloud Server Provider*
EDA	*Estimation of Distribution Algorithm*
FIFO	*First In First Out*
FHE	*Full Homomorphic Encryption*
GA	*Genetic Algorithm*
IoT	*Internet of Things*
MVC	*Model View Controller*
NCP	Plug-in Nagios-Ceilometer
QoS	*Quality of Services*
SO	*Synthesis Optimization*
TAP	*Trusted Auditing Proxy*
TI	Tecnologia da Informação
VB6	*Visual Basic 6*
VBUC	*Visual Basic Upgrade Companion*
VM	*Virtual Machine*
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

A computação em nuvem é uma tendência recente de tecnologia que tem por objetivo proporcionar serviços de Tecnologia da Informação sob demanda com pagamento baseado no uso destes serviços (RUSCHEL;ZANOTTO;MOTA, 2010). Em termos práticos, a nuvem é a capacidade ociosa de servidores de larga escala, como os disponibilizados por Google e Microsoft, que pode ser emprestada ou vendida a quem julgar necessário, podendo assim guardar ou processar seus volumes de dados, além de executar suas computações (ALVES;COSTA;FURTADO, 2017). A computação em nuvem traz imensa versatilidade e elasticidade, além de oferecer escalabilidade e significativo poder de distribuição.

Quando fala-se em sistemas, de modo geral, pensa-se em algo complexo e de imensa importância para as empresas que, na falta destes, teriam suas atividades inviabilizadas. Na maioria dos casos os sistemas rodam localmente, sendo necessário manter uma infraestrutura de servidores para a execução destes. Além da execução dos sistemas, é comum que estes mesmos servidores mantenham o armazenamento de todas as suas informações relacionadas aos mesmos.

Para se montar um servidor responsável pela execução de um sistema, onde a empresa ficará dependente deste servidor, presume-se que deve-se tomar cuidado ao escolher o hardware e os profissionais que tomarão conta do mesmo. Para suprir esta necessidade, torna-se obrigatório um investimento inicial geralmente significativo, gerando custos para a empresa. Estes custos referem-se tanto ao *hardware* quanto ao *software*, além de demandar também recursos humanos e mais toda a infra-estrutura necessária para se manter um parque de servidores, como *no-breaks*, *switches*, e *storages*.

Com o passar do tempo, um parque de servidores pode sofrer problemas, tornando necessário manter equipamentos redundantes que permitam prevenir possíveis falhas, gerando ainda mais custos e perda de produtividade. Em geral, é necessário parar os sistemas durante seu período de manutenção, podendo inclusive inviabilizar atividades da empresa. Pela ótica da propriedade do ativo físico, é importante destacar que todo o *hardware* tende a ficar obsoleto, sendo necessária a aquisição de novos equipamentos.

Substituir um sistema pode ser um problema gigantesco já que, além do esforço em se instalar um novo sistema, é necessário configurá-lo por completo, importar todo o histórico do sistema anterior, além de treinar os usuários. Este processo é trabalhoso e complexo, não

só pelo trabalho do pessoal da TI, mas também por todos os usuários que terão de se adaptar com este novo sistema. Ao calcular todos os custos e riscos envolvidos para manter servidores locais, mesmo desprezando custos com energia elétrica, manutenções de *hardware*, espaço de armazenamento limitado, pode-se concluir que esta não é uma opção viável para parte das empresas. Atualmente, servidores totalmente online dispensam a compra de qualquer equipamento de *hardware*, estando disponíveis constantemente na internet. Uma conexão remota é o suficiente para ter acesso a este modelo de servidores.

A migração de sistemas legados de uma organização para ambientes de computação em nuvem, representa grandes vantagens na análise de custo-benefício (por exemplo, pague apenas pelo que é usado, menor investimento em hardware e manutenção técnica, etc.) (ZALAZAR;GONNET;LEONE, 2015).

A eliminação da necessidade de manter um servidor local, como já abordado, traz uma série de vantagens competitivas para uma empresa. Dentre estas vantagens elenca-se:

I. Não é preciso arcar com o custo de compra do *hardware* (servidor, *storages*, switches);
II. O *hardware* não ficará obsoleto para o consumidor, já que quando de sua obsolescência, pode-se configurar um equipamento novo em poucos instantes, atualizando os recursos;
III. Não há custo de energia elétrica extra, visto que o servidor não estará nas dependências da empresa, não consumindo seus recursos;
IV. É possível reduzir os custos configurando os servidores contratados para que sejam desligados em períodos de inatividade (especialmente na madrugada);

A migração do sistema fornece a capacidade de integrar alguns aplicativos em um solução única de software, permitindo criar processos colaborativos entre clientes, parceiros e diferentes fornecedores (MEZGAR; RAUSCHECKER, 2014).

Dado que as vantagens são inúmeras, visto que manutenções, atualizações e custos com energia são desprezíveis, pode-se manter o foco naquilo que é fundamental para a continuidade das atividades da empresa: o sistema. Como citado, sabe-se que uma troca de sistema representa um problema. Por outro lado, se fosse possível converter um sistema legado em um sistema novo, atrelando este a um servidor em nuvem, mantendo todas as suas funcionalidades, aumentando sua disponibilidade e elasticidade, chegaria-se a um cenário

ideal. Nesta migração poderiam ser revistos também aspectos visuais, já que as ferramentas atuais tendem a apresentar um aspecto mais adequado às tendências, além de serem compatíveis com interfaces Web. Hoje em dia existem ferramentas capazes de construir sites responsivos, sendo possível visualizá-los tanto em smartphones quanto em tablets ou desktops.

Baseado no cenário exposto, este trabalho realizou um estudo a fim de superar as dificuldades e impedimentos, permitindo o desenvolvimento de um *framework* para conversão de aplicações legadas em aplicações Web. Para tal, foram criados componentes dinâmicos com o intuito de ampliar a compatibilidade do referido *framework* com um maior número de projetos legados.

Este trabalho está estruturado da seguinte maneira: o Capítulo 2 apresenta Trabalhos Relacionados, tendo o objetivo de buscar informações semelhantes ao trabalho proposto, e conta com uma seção de discussão, que realiza a justificativa dos mesmos.

O Capítulo 3 apresenta o desenvolvimento do *framework*, bem como uma seção designada a contribuição com a comunidade.

No Capítulo 4 são apresentados os experimentos e seus resultados, sendo demonstrado como foram realizados os testes do *framework,* descrevendo um questionário de pesquisa que foi aplicado aos usuários finais.

O Capítulo 5 apresenta conclusões e apontamentos de trabalhos futuros.

2 TRABALHOS RELACIONADOS

Esta seção trará alguns trabalhos relacionados ao tema proposto, abordando assuntos tangenciais à conversão de sistemas para nuvem.

Para encontrar estes trabalhos relacionados, foi realizada uma revisão sistemática no assunto de conversão de sistemas para nuvem, a fim de descobrir quais as técnicas mais empregadas, para isso foram consultadas 3 bases de dados:

- IEEE;
- Google Academic;
- Web of Science;

Foi realizada a pesquisa nas bases de dados com o texto de busca ((“*convert*” *OR* (“*development*”)) *AND* (“*system*”) *AND* (“*cloud*”), e foram retornados 337 artigos ao todo.

Destes artigos retornados, foram analisados os títulos e resumos a fim de se realizar um primeiro refino, com isso foram selecionados 22 artigos que mais se relacionam com o tema proposto.

No segundo refino, foi efetuada a leitura das introduções e conclusões dos 22 artigos. Após esta leitura, restaram 5 artigos que foram lidos na íntegra, e os mesmos serão explanados nas seções a seguir.

2.1 CLOUDPROCMON: A NON-INTRUSIVE CLOUD MONITORING FRAMEWORK

A computação em nuvem fornece serviços sob demanda. Atualmente as empresas não precisam investir grande capital em infraestrutura de TI e pessoal qualificado para cuidar dele, ao invés disso, elas podem obter serviços de TI usando a computação em nuvem. As nuvens são grandes sistemas distribuídos responsáveis pela execução de serviços, mas para saber se o desempenho está como esperado, é necessário um monitoramento (SYED; GANI; NASARUDDIN, 2018).

O monitoramento da nuvem requer uma coleta de informações, e essa coleta pode ser dividida em dois tipos principais: monitoramento intrusivo e monitoramento não intrusivo. O monitoramento intrusivo utiliza um softwares nas VMs para colher informações, e o monitoramento não intrusivo não utiliza software, coletando os dados a partir do sistema operacional *host.* Syed, Gani, Nasaruddin, et al. propuseram o desenvolvimento de um

framework de baixo consumo de recursos, que será responsável pela captura das informações de monitoramento, de forma que haja uma sobrecarga insignificante no servidor *host*, e ainda possuindo alta escalabilidade.

O CloudProcMon (*framework* proposto), realiza a coleta de dados no sistema operacional *host*, consultando arquivos do *Kernel*, sendo possível obter o estado atual de um processo. O "Procfs" é um arquivo pseudo virtual, que fornece estatísticas de processos em execução. Para executar o monitoramento não intrusivo, alguns dados métricos são coletados deste arquivo. Três algoritmos são responsáveis pela coleta das informações de CPU, rede e memória. O algoritmo 1 é responsável por abstrair as informações do arquivo "Procfs", referentes ao uso de CPU, o algoritmo 2 é responsável pelas informações de rede e o algoritmo 3 é responsável pelas informações de memória.

Para saber se o *framework* proposto obteve resultados, foi adotado outro *software* semelhante, para comparar os resultados. O CloudProcMon foi comparado com o *Plug-in Nagios-Ceilometer* (NCP) (BENZ. 2014). Para realizar a comparação de desempenho, foram criados ambientes semelhantes, e os fatores de aferição foram latência de monitoramento, frequência de pesquisa e sobrecarga.

Após os testes de monitoramento de latência, o CloudProcMon conseguiu realizar o monitoramento em tempo real em apenas 3,45 segundos, comparados com 11,22 segundos do NCP, provando que o CloudProcMon é mais eficiente que o NCP.

A próxima etapa foi realizar os testes de frequência de pesquisa. A frequência de pesquisa é um recurso importante, uma frequência muito alta pode criar uma sobrecarga insuportável para um sistema em nuvem, e uma frequência muito baixa será inútil. Após efetuar uma investigação mais profunda no NCP, foi descoberto que o mesmo tem uma taxa de coleta padrão de 10 minutos, enquanto o CloudProcMon leva 4 ms para buscar os dados, provando que CloudProcMon dá resultados precisos sem causar nenhuma sobrecarga no sistema.

Para finalizar os testes, foi realizado um monitoramento de sobrecarga comparando o sistema com e sem as ferramentas. O NCP teve uma sobrecarga de 10 a 15% para realizar o monitoramento, considerado bastante alto. E o CloudProcMon teve uma sobrecarga de 2 a 4% para realizar o monitoramento, demonstrando que a sobrecarga é praticamente insignificante.

Os resultados foram muito satisfatórios, e fornecendo evidências convincentes que o *framework* CloudProcMon consegue monitorar um sistema em nuvem de forma não intrusiva,

e com uma sobrecarga insignificante. Contudo, a seguinte limitação é importante notar, embora as hipóteses fossem suportadas estatisticamente, apenas três métricas foram coletadas como prova de conceito. Para trabalhos futuros, o autor pretende adicionar novas métricas de investigação, empregando o *framework* CloudProcMon para avaliar o sistema em nuvem.

2.2 CONTEXT-AWARE VERIFIABLE CLOUD COMPUTING

A internet das coisas (IoT) fornece serviços avançados e inteligentes para os seres humanos. Ela está evoluindo de maneira importante para o próximo paradigma de rede e infraestrutura. A computação em nuvem fornece uma grande variedade de recursos ao se conectar a serviços de rede, e também pode trabalhar em conjunto com a IoT para fornecer serviços de computação que liberam o processamento de big data em dispositivos.

O *Cloud Server Provider* (CSP) não é totalmente confiável e a privacidade das informações devem ser preservadas, os dados coletados são criptografados, para processamento posterior. Um smartphone pode coletar inúmeros dados para serem processados e posteriormente utilizados em serviços de IoT para oferecer serviços inteligentes.

Yan, Yu e Ding propuseram um esquema de verificação da computação, com consciência do contexto e preservação da privacidade em computação em nuvem IoT. Aplicando *Full Homomorphic Encryption* (FHE) para processar os dados criptografados no CSP, e o *Trusted Auditing Proxy* (TAP) para a auditoria.

Para alcançar um processamento de dados confiável e evitar um potencial risco aos serviços em nuvem, o projeto deve atingir as seguintes metas:

- Segurança e proteção;
- Generalidade;
- Sobrecarga leve;

Foi avaliado o desempenho de 4 protocolos, comparando e avaliando sua segurança, complexidade computacional, custo de comunicação e escalabilidade. O protocolo 1 foi o que obteve a menor segurança, a menor sobrecarga, menor custo de comunicação e a melhor escalabilidade. O protocolo 2 obteve a média em todos os pontos de avaliação. O Protocolo 3 obteve uma excelente segurança, uma alta sobrecarga, um custo de comunicação médio alto, e

a pior escalabilidade. O protocolo 4 também obteve uma excelente segurança, uma sobrecarga média, o maior custo de comunicação e uma escalabilidade média.

Com base nessas informações, Yan, Yu e Ding recomendam a utilização do protocolo 1 para quando a aplicação não exige uma alta segurança, visto que pode atingir um bom desempenho relacionado a sobrecarga, comunicação e escalabilidade. Caso a segurança seja um requisito de alta importância, o protocolo 4 entrega um bom custo computacional e uma boa escalabilidade.

Os autores propuseram com este artigo verificar se os protocolos de auditorias podem aumentar a confiança da computação em nuvem. Com os estudos realizados foi possível identificar que os protocolos de auditoria propostos, podem ajudar a verificar se a integridade e o processamento dos dados no CSP são satisfatórios. Além do esquema proposto servir como estrutura genérica para apoiar a computação verificável de diferentes contextos. Quatro protocolos de auditoria foram projetados para fornecer diferentes requisitos de segurança. Seu desempenho foi analisado em relação a segurança, sobrecarga, custo computacional e escalabilidade. Os resultados mostraram eficácia e eficiência, por meio de uma comparação rigorosa com base na sua aplicabilidade.

2.3 AN EDA-GA HYBRID ALGORITHM FOR MULTI-OBJECTIVE TASK SCHEDULING IN CLOUD COMPUTING

A computação em nuvem fornece recursos de hardware e software na modalidade pague conforme o uso. Nesta modalidade os usuários podem utilizar os serviços sem a necessidade da compra de toda infraestrutura e/ou softwares. Como os serviços crescem rapidamente, o agendamento de novas tarefas se tornou muito importante. Quanto mais rápido o serviço for executado, melhor será a experiência do usuário e menor será o desperdício de recursos por ociosidade. Mas não é tão simples quanto parece, pois se agendar todo o processamento para executar determinada tarefa, ocorrerá um desbalanceamento das cargas. Otimizar a execução das tarefas para a conclusão em menor tempo e ainda melhorar o balanceamento das cargas é um desafio.

Pang, Li, He et al. propuseram um modelo de agendamentos multi-objetivo detalhado e um algoritmo híbrido entre EDA - GA para resolver o agendamento de tarefas multi-objetivo.

EDA é um algoritmo evolutivo baseado em população, eficaz em problemas de otimização (LARRAANAGA, LOZANO. 2001; SUN, ZHANG, TSANG. 2005), enquanto GA é um algoritmo que simula a seleção natural, e é mais utilizado em pesquisas globais (OMARA, ARAFA. 2010; YU, BUYYA. 2006).

O algoritmo híbrido EDA-GA é projetado da seguinte maneira: primeiramente se utiliza o EDA para iniciar o modelo de probabilidade. Segundo, se utiliza GA para realizar operações cruzadas e modificar as soluções selecionadas. Terceiro, avalia-se as soluções excelentes da etapa 1 e as novas soluções da etapa 2, classificando-as em ordem decrescente. Por último, atualize o modelo de probabilidade. Execute o algoritmo até a solução de parada for atendida e gerar a solução ideal.

Para testar a eficiência do algoritmo híbrido EDA-GA, foi comparado EDA com GA com base no CloudSim. CloudSim é um software de simulação de computação em nuvem anunciado pela *Grid Laboratory* da *University of Melbourne* e Projeto Gridbus em abril de 2009. O objetivo principal é comparar a estratégia de programação entre diferentes modelos de aplicativos.

Nos testes práticos, foram criadas 3 instâncias diferentes: Instância 1, pequenas tarefas junto com muitas tarefas grandes. Instância 2, algumas tarefas grandes junto com muitas tarefas pequenas. Instância 3, os tamanhos das tarefas são aleatórias. Foram 1000 tarefas e 10 máquinas virtuais para as 3 instâncias. Foram observados nos resultados o tempo de conclusão, o balanceamento de carga e o valor *fitness*. Valor *fitness* é usado para avaliar a quantidade de soluções, evitando cair em um ótimo local para alcançar uma solução ideal.

Ao término dos testes, os resultados mostraram que EDA-GA possui um tempo de conclusão da tarefa menor, uma carga mais equilibrada e maior valor *fitness.* Além disso, as operações cruzadas e as modificações, expandem o leque de soluções para evitar o algoritmo de cair em um ótimo local, o algoritmo híbrido EDA-GA possui o melhor desempenho.

Os resultados mostraram que o algoritmo híbrido EDA-GA proposto, tem boa velocidade de convergência e grande poder de pesquisa, conclui as tarefas em menor tempo e melhora o balanceamento de carga.

2.4 DYNAMIC PERFORMANCE OPTIMIZATION FOR CLOUD COMPUTING USING M/M/M QUEUEING SYSTEM

A computação em nuvem é um recurso que permite o acesso de qualquer lugar conveniente, sob demanda, e podem ser rapidamente provisionados. De modo geral, a computação em nuvem fornece recursos escalonáveis em tempo real, como hardware e software. Em todos os aspectos o *Quality of Services* (QoS) é a base da computação em nuvem já que inclui disponibilidade, rendimento, confiabilidade e segurança.

Guo, Yan, Zhao et al. propuseram um modo otimizado de fila, por meio da teoria de filas M/M/m (KLEINROCK. 1975), projetaram uma função de otimização sintética, resultando em uma melhora de desempenho do servidor em comparação com o método clássico, menor serviço primeiro e primeiro a entrar, primeiro a sair.

Neste artigo, o servidor é moldado como um serviço central, que pode ser utilizado como fila para serviços, com chegadas múltiplas de tarefas e buffer de solicitação com capacidade para infinitas tarefas. Este modelo de servidor consiste em um centro de serviços. Centro de serviços é um ponto de acesso único para todos os clientes. É uma coleção de recursos que hospeda os serviços e aplicativos para usuários.

Supondo que existam várias solicitações e vários serviços, cada um deles é independente, o tempo de recebimento das requisições é aleatório, e podem ser de mais do que um usuário solicitando. A estratégia de otimização aplicada foi: todas as solicitações que chegam ao agendador passam pela função de otimização de resultado. Os parâmetros de entrada da função de otimização seriam tempo médio, utilização e total de clientes esperando uma resposta do servidor.

A validação do desempenho da estratégia de otimização se deu através do software MATLAB (software voltado a cálculos numéricos) versão R2009b. O método adotado foi a fila prioritária de *synthesis optimization* (SO).

Para comparar e analisar as estratégias de otimização, foi utilizado fila de prioridade clássica, menor serviço primeiro, com política de fila primeiro entrar primeiro a sair (FIFO). Foi comparado o desempenho de duas políticas de fila, priorizadas com FIFO.

Foi testando o tempo médio de espera dos servidores. Quando o servidor possui serviços pequenos, o tempo médio de espera é mínimo, no entanto quando o serviço aumenta, a política do SO otimiza a utilização, tempo de espera e comprimento médio da fila, tirando o melhor proveito de cada serviço, tornando o tempo médio na fila menor.

Neste artigo os autores propuseram uma fila modelo e desenvolveram um método de otimização sintético para otimizar o desempenho da execução de serviços. O resultado da

simulação mostrou que é possível otimizar as filas de serviços com o método proposto, gerando menos tempo de espera, menor comprimento de fila e maior número de clientes utilizando o serviço.

2.5 EVOLUTION OF CLOUD OPERATING SYSTEM: FROM TECHNOLOGY TO ECOSYSTEM

Muitos aplicativos tradicionais estão migrando para sistemas em nuvem. Desenvolvedores e provedores de serviço não precisam se preocupar com a construção da infraestrutura, atualização de hardware ou manutenção de servidores. A plataforma para execução de aplicativos em nuvem pode ser chamada de sistema operacional, um sistema operacional em nuvem serve para gerenciar recursos distribuídos em grande escala, semelhante ao sistema operacional tradicional, gerenciando o hardware em uma única máquina. Os aplicativos em nuvem podem ser de diferentes formatos, como micro serviços, processamento em lote (ISARD, BUDIU, YU, et al. 2007; ZAHARIA, CHOWDHURY, DAS et al. 2012; POWER, LI. 2010) e processamento de streaming de dados (NEUMEYER, ROBBINS, NAIR et al. 2010; VIGLAS, NAUGHTON. 2002; SHEN, ZHANG. 2008).

O sistema operacional em nuvem está evoluindo de geração em geração, e a evolução dos sistemas operacionais em nuvem se dá a dois objetivos considerados conflitantes, melhoria de eficiência e melhoria de experiência, pois melhor eficiência implica em menor experiência e uma melhor experiência implica em uma menor eficiência.

Zuo-Ning Chen et al propuseram neste artigo estudar as formas de sistemas em nuvem considerando três aspectos diferentes: evolução da tecnologia, evolução de arquitetura do sistema e a evolução do ecossistema da nuvem.

As APIs (*application programming interfaces*) estão tendo um papel muito importante nos sistemas em nuvem, pelo fato delas estarem cada vez mais estáveis, mais interfaces estão sendo adicionados nelas, e às vezes projetos diferentes utilizam a mesma API, facilitando a manutenção e implementação.

O desempenho quase sempre é a principal preocupação em relação à evolução da computação, e a melhora do desempenho da computação em nuvem não é diferente. Basicamente existem duas maneiras de melhorar o desempenho de sistemas em nuvem: primeira maneira seria estar sempre com o último hardware lançado, e a outra maneira seria

por meio de aprimoramento do software. Durante a evolução dos sistemas em nuvem houve muitos avanços de hardware, principalmente para a melhora do desempenho. Além disso, a topologia de rede de um servidor é muito importante para o desempenho da computação em nuvem (GREENBERG, HAMILTON, JAIN et al. 2009; PARAISO, HADERER, MERLE et al. 2012). Além da evolução e melhoria no desempenho do hardware dos sistemas em nuvem, outro ponto importante que também evoluiu é a programação.

Nos últimos anos, o surgimento de contêineres melhorou o desenvolvimento dos sistemas em nuvem. Os contêineres isolam uma parte do sistema, reduzindo o número de abstrações, melhorando a eficiência. Agora a ferramenta de gerenciamento de containers Docker (HARTER, SALMON, LIU et al. 2016) se torna muito popular.

Além das tecnologias mencionadas, também foi investigado a questão dos testes de sistema, com o objetivo de efetuar uma avaliação completa de todo o sistema em termos de complexidade da API e desempenho da interface. A análise da avaliação resultou em um esquema de teste automatizado baseado na semântica da interface, protegendo diferentes interfaces e reduzindo o custo computacional do teste.

Os autores apresentaram neste artigo um breve estudo da evolução das tecnologias relacionadas aos sistemas operacionais em nuvem, e também salientam que as APIs possuem um papel fundamental em sistemas em nuvem. Bem gerenciado os recursos, os sistemas em nuvem podem se tornar sistemas bastante produtivos e saudáveis.

2.6 FERRAMENTAIS PARA ABORDAGENS PRÁTICAS

Agora nesta seção serão apresentados alguns *frameworks* relacionados ao tema proposto.

2.6.1 THINFINITY VIRTUAL UI

Thinfinity Virtual UI é uma plataforma que permite qualquer aplicativo Windows ser publicado como uma solução Web, bastando inserir apenas uma linha de código ao código-fonte da aplicação.

A plataforma desenvolvida pela Cybele Software é muito interessante, pois leva consigo um *framework* capaz de emular a aplicação em uma página web. Para se utilizar esta

ferramenta, deve-se baixar o *framework* disponibilizado, procedendo com a instalação e configuração da licença do produto. A Cybele Software disponibiliza licença para teste com a duração de 30 dias. Também no site do fabricante, existe uma área voltada à tutoriais, com diversos projetos de diversas linguagens.

Foi efetuado o *download* de um projeto na linguagem C# WinForms para a realização do teste. Se obteve dificuldade para funcionar a plataforma, pois não foi localizada informações de como proceder com a ativação da licença de teste. Após concluir a ativação, foi executado o projeto, e o mesmo ao invés de rodar localmente, abriu uma nova janela no browser com a aplicação, mas após alguns segundos a aplicação também foi executada localmente. Logo, foi notado alguns problemas ao utilizar a aplicação. A aplicação que estava sendo executada localmente conflitava com a que estava rodando no browser. As duas aplicações permaneciam espelhadas, causando alguns problemas de usabilidade. Também não foi possível abrir a aplicação em mais de um browser. Ao abrir em um segundo browser, o primeiro era encerrado instantaneamente, mas isto acredita-se ser uma limitação da licença, pois em vídeos de demonstração, era possível abrir a mesma aplicação em diversas abas do browser e inclusive outras instâncias de browsers.

Por mais que seja muito prático apenas inserir uma linha de código para ter este recurso, é necessário comprar a licença para poder usufruir desta funcionalidade, e por mais que esteja na Web, o *design* da aplicação continuará sendo um *design* de uma aplicação WinForms. Além de não ser responsiva, forçando o usuário a utilizar em apenas dispositivos com uma tela grande, pois em telas pequenas como as dos smartphones, a tela da aplicação poderá quebrar ou até mesmo aparecer algum *scroll* vertical e ou horizontal na tela, para poder visualizar por completo as informações da aplicação, tornando a experiência inadequada.

Não foram realizados testes quanto a impressão de relatórios, pois o projeto no qual foi realizado o teste não havia disposição de relatórios. Não foi visto nem na apresentação da ferramenta como ficaria a impressão de relatórios. No entanto, no WinForms a impressão acontece graças a um recurso da ferramenta de relatórios empregada na aplicação, já na Web a impressão acontece através do browser, imprimindo diretamente o html da página, ou através da instalação de algum *plugin* para impressão de relatórios, normalmente arquivos com extensão “.pdf”.

Para se ter uma real experiência da plataforma, com todos os seus recursos disponíveis, é necessário a compra da licença.

2.6.2 WEBMAP

O WebMap é uma ferramenta de migração de códigos legados para a Web. Ela é capaz de converter aplicações WinForms C# para ASP.NET Core, mas nos exemplos demonstrados em vídeo, somente é possível a conversão a partir do WinForms, visto que no exemplo do vídeo, a aplicação estava na linguagem VB6, e primeiramente foi convertido para WinForms C#, e posteriormente convertido para ASP.NET Core.

A Mobilize.net, empresa responsável pela aplicação WebMap, não disponibiliza nenhuma versão para teste ou tutorial de execução. Com base nesta percepção, toda análise aqui descrita refere-se aos vídeos disponíveis no site da empresa. As opções de *download* no site da Mobilize.net são apenas de um *software* de análise do código fonte que apenas lista todos os arquivos do projeto analisado, demonstrando a quantidade de linhas, tamanho dos arquivos, e mostra os resultados em um gráfico.

Já a outra opção de *download* é de uma ferramenta para conversão de código VB6 para WinForms C# ou VB.NET, na versão de avaliação, chamada de *Visual Basic Upgrade Companion* (VBUC), que por ser versão de avaliação tem restrições quanto ao uso, semelhantemente a outra plataforma analisada. Esta plataforma permite, em versão de avaliação, a conversão de dez mil linhas de código que se refletem em projetos médios, caso o usuário queira testar diretamente em sua própria aplicação.

Quanto a conversão de WinForms para ASP.NET Core, foi percebido que nas apresentações em vídeo, ao converter a aplicação, era mantido o *design* das telas o mais próximo do original, sem nenhuma inovação quanto ao *design.* Do ponto de vista técnico, acredita-se ser o mais fácil a se fazer, pois quanto mais próximo do modelo original, menor será o trabalho de comparar se a conversão obteve sucesso. Do ponto de vista comercial, já que se está convertendo uma aplicação com *design* antigo e componentes antigos, para Web, espera-se ao menos alguma inovação no *design*, pois atualmente existem componentes responsivos e mais elegantes, mas isto deve impactar no tempo de desenvolvimento e um aumento significativo da complexidade do projeto.

Nos vídeos de demonstração do WebMap, também não foram demonstradas nenhuma conversão de possíveis relatórios, semelhante ao observado no Thinfinity Virtual UI.

2.7 DISCUSSÃO

Cada trabalho relacionado aqui demonstrado, possui uma ligação com o trabalho proposto. Na seção 2.1 o trabalho fala de monitoramento intrusivo e não intrusivo, e explica como foi realizado o monitoramento dos recursos de uma VM, de maneira intrusiva, instalando um software para captar essas informações ou captando as informações diretamente do sistema operacional do *host*.

A relação com o trabalho proposto está justamente em abordar a importância de saber qual o consumo de recursos que a aplicação está recrutando, assim sendo possível observar ao longo do tempo, se a aplicação convertida utiliza mais ou menos recursos do sistema operacional, comparado com a aplicação original.

Na seção 2.2 o trabalho menciona a importância da privacidade das informações, reconhecendo que um CSP não é seguro, e precisa criptografar as informações para manter sigilo. O trabalho proposto foi montado em uma plataforma atual e altamente tecnológica, possibilitando manter o sigilo das informações, bem como sugere esta seção.

A seção 2.3 e a seção 2.4 trazem o tema de escalonamento de processos e otimização de filas, buscando uma maneira de otimizar ao máximo o processamento das tarefas, e a relação com o trabalho proposto está justamente na escolha da plataforma, por ser uma tecnologia atual, ela consistem em atender as demandas atuais, visando performance, segurança e elasticidade (MICROSOFT, 2020b).

Na seção 2.5 o artigo comenta a evolução da computação em nuvem, recursos que estão em constante evolução e aprimoramento, sistemas operacionais mais capazes de atender a demanda de processamento, APIs focadas no tráfego de dados. Com base nisto, a relação com o trabalho proposto é justamente a evolução de uma tecnologia antiga para uma nova, a evolução dos sistemas, com o objetivo de melhorar a compatibilidade da aplicação antiga com as novas plataformas presentes na atualidade.

A seção 2.6 teve o objetivo de pesquisar por *frameworks* relacionados, pois antes de dar início neste projeto, foi necessário verificar a existência de aplicações compatíveis com a

deste trabalho, a fim de verificar se o mercado já havia solucionado este problema relacionado às aplicações legadas.

Foram localizadas duas aplicações, mas apenas a aplicação WebMAP realiza a conversão do projeto legado para um projeto novo, já a aplicação Thinfinity Virtual UI realiza uma emulação da aplicação legada na Web, mantendo toda a aplicação sem alterações.

O uso destas duas ferramentas variam de acordo com a necessidade de cada empresa, pois a ferramenta que faz a emulação do projeto legado e não requer muitas modificações, segundo Thinfinity Virtual UI, basta informar uma linha de código no código fonte para a aplicação ser emulada para a Web.

Já a ferramenta WebMAP, promete converter o projeto inteiro para um novo projeto, bem como a ideia deste trabalho, pois deste modo é possível usufruir dos recursos que a nova plataforma terá disponível, bem como modificar interfaces e componentes.

3 DESENVOLVIMENTO

Dando início ao desenvolvimento do *framework*, no primeiro momento foi feita a análise do projeto legado, a fim de se mapear quais seriam os impedimentos ou atrasos devido a dificuldade em se desenvolver um código dinâmico que fosse capaz de abranger os futuros projetos que poderão ser convertidos com esta ferramenta.

Como esperado, foram encontradas algumas dificuldades no que diz respeito à captação de referências do projeto legado. As referências de um projeto são elementos de software como serviços, pacotes, bibliotecas ou APIs utilizadas pelos *frameworks* referenciados (MICROSOFT, 2019a).

Um projeto às vezes pode possuir inúmeras referências, podendo variar entre projetos da própria empresa e projetos de terceiros, aos quais não se tem acesso ao código fonte, tornando o desenvolvimento ainda mais desafiador. O problema é que dependendo da tecnologia utilizada nesta referência, pode se resultar em algum erro por alguma possível incompatibilidade do *framework* atual com o *framework* utilizado nas dependências da referência.

Posteriormente foi criado um projeto modelo em ASP.NET Core 5.0 MVC (MICROSOFT, 2020a), no qual constituirá a base da conversão do projeto legado. Logo após a criação do projeto, o desenvolvimento andou de forma paralela com a análise do projeto a ser convertido.

A tarefa de mapeamento dos componentes utilizados no projeto legado, e a escolha dos novos componentes, foram construídos em paralelo no novo projeto, desta forma deixando o código mais limpo, visto que não será necessário escrever o código de cada componente sempre que o componente equivalente no projeto legado aparecer em um form.

Agora que os componentes estão modelados, quando for necessário utilizar um componente, bastará copiar este componente do projeto modelo, realizar as substituições necessárias como por exemplo, nome, posicionamento, eventos e demais eventos presentes no form, e o restante dos códigos necessários, como, *controller*, *model*, *interface* e qualquer outra classe que seja necessária, e a gravação dos mesmos no local adequado de acordo com a nova tecnologia.

Por se tratar de uma conversão de uma tecnologia antiga para uma nova, e considerando as boas práticas (MICROSOFT. 2020b), haverá mudanças na estrutura de pastas e arquivos de dentro do projeto, com o intuito de construir um *framework* que seja capaz de atender os pré-requisitos descritos aqui neste trabalho.

3.1 PROJETO MODELO

O projeto modelo consiste em desenvolver um elemento de software que será utilizado como base da conversão do projeto legado. Este projeto foi construído em ASP.NET Core 5.0 MVC (1-MICROSOFT, 2020) com Razor páginas (MICROSOFT, 2020c), e também serão consumidas algumas bibliotecas como:

- DataTables (DATATABLES, 2021);
- Bootstrap (BOOTSTRAP, 2021);
- JavaScript (MDN, 2021b);
- Jquery-inputmask (JQUERY, 2014);
- Jquery-ajax (JQUERY, 2021);
- Docker (DOCKER, 2021);

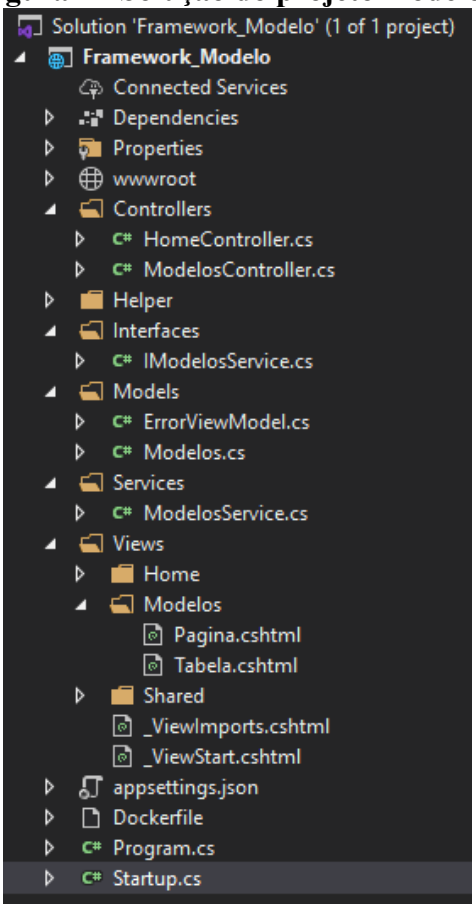
Este projeto ficará embutido na aplicação que o desenvolvedor irá interagir para gerar o projeto convertido com este projeto servindo de base.

Como pode-se observar na Figura 1, por se tratar de um projeto MVC, existem vários arquivos que compõem o projeto modelo, sendo os arquivos principais:

- ModelosController.cs;
- IModelosService.cs;
- Modelos.cs;
- ModelosService.cs;
- Pagina.cshtml;
- Tabela.cshtml;

Estes arquivos serão renomeados conforme o nome da tela que o originará, bem como a clonagem dos mesmos caso seja necessário.

Figura 1 - Solução do projeto modelo



Fonte: Visual Studio 2019

3.2 *FRAMEWORK* - APLICAÇÃO

A aplicação consiste em criar um projeto que será responsável pela interação com o desenvolvedor, que irá acompanhar a conversão do projeto legado e posteriormente efetuar ajustes de acordo com a necessidade.

Esta aplicação será desenvolvida em WinForms. Foi realizada uma análise e chegou-se a conclusão que não seria necessário o desenvolvimento desta aplicação para a plataforma Web. Por se tratar de uma ferramenta de conversão que lê arquivos de um projeto, cujo projeto legado normalmente se encontra em um servidor da empresa, julga-se um uso mais amigável para o desenvolvedor uma aplicação de instância local.

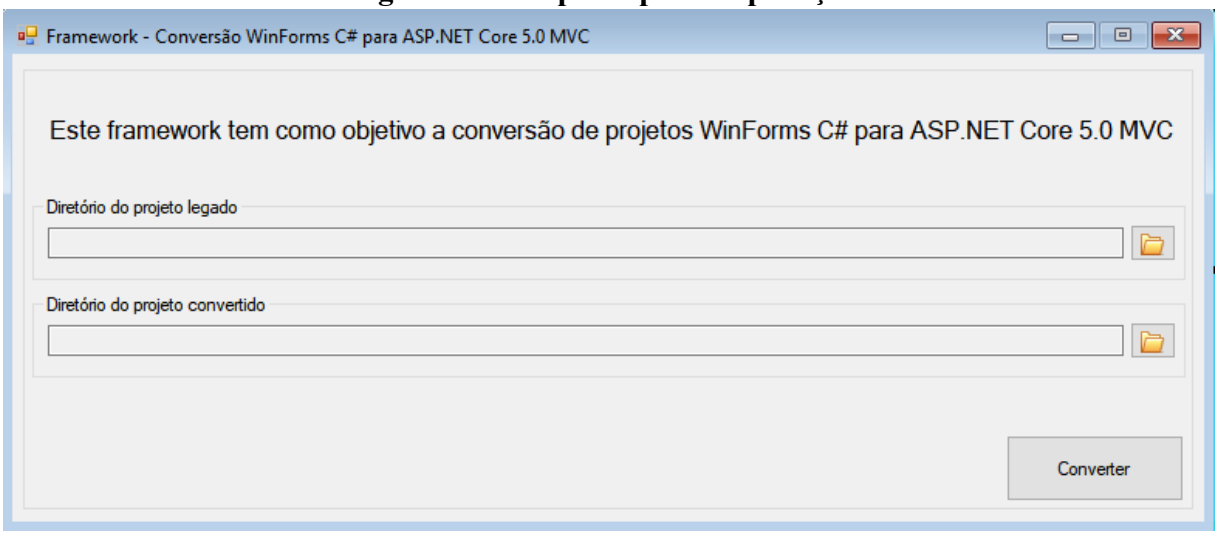
Por se tratar de uma tarefa complexa, é sabido que será necessário algum ajuste no projeto após a conversão, desde uma simples troca de cor de fonte a algum possível problema mais complexo, sendo necessário algum ajuste por parte do desenvolvedor.

Conforme a Figura 2, esta aplicação exige que sejam informados alguns parâmetros, como:

- Diretório do projeto legado;
- Diretório do projeto convertido;

Com base nestes parâmetros que serão informados pelo desenvolvedor, será feita a conversão do projeto legado.

Figura 2 - Tela principal da aplicação



Fonte: Elaborado pelo autor (2021).

As etapas da conversão estão dispostas na seguinte ordem:

- Clonando o projeto modelo;
- Copiando estrutura de classes do projeto legado;
- Localizando as referências do projeto legado;
- Convertendo os forms;

Todas estas etapas serão explanadas nas próximas seções.

3.3 CLONANDO O PROJETO MODELO

A primeira etapa é a clonagem do projeto modelo, dando início à conversão. Esta clonagem acontece após o desenvolvedor informar os parâmetros base mencionados anteriormente na Figura 2, e clicar no botão “Converter”, quando é feita a cópia do projeto modelo para o diretório escolhido pelo desenvolvedor. Esta cópia, em primeiro momento, não sofre nenhuma modificação.

3.4 COPIANDO A ESTRUTURA

Nesta etapa será realizada a cópia da estrutura de pastas e classes do projeto legado para dentro do projeto modelo que foi clonado. É interessante salientar que existem algumas restrições quanto a cópia de determinados arquivos. O arquivo “program.cs” não será copiado, pois este arquivo é utilizado para iniciar a aplicação e também normalmente utilizado para algumas configurações específicas, mas ele será consultado no decorrer da conversão.

Na estrutura do projeto, os demais arquivos com as extensões: “.csproj”, “.user”, “.vspscc”, “.sln” e “.config” também não serão copiados, pois eles constituem as configurações, às referências, às preferências, e as mesmas estão constituídas em outros arquivos no novo projeto. Semelhantemente ao arquivo “program.cs”, estes arquivos cujo as extensões são iguais às mencionadas, serão consultados no decorrer da conversão a fim de levar o máximo possível das configurações.

Algumas pastas também não serão copiadas para o novo projeto, visto que duas delas (bin, obj) são geradas sempre que o projeto é compilado, e para a conversão elas não apresentam relevância. A pasta “Properties” contém propriedades relacionadas ao projeto referente ao Visual Studio e não serão consideradas para o novo projeto.

Na Figura 3 consta o método criado para realizar a cópia da estrutura e classes, como descritas aqui nesta seção.

Figura 3 - Método responsável por efetuar a cópia das pastas e classes do projeto legado

```
public static void Estrutura()
{
    string[] files = Directory.GetFiles(Parametros.DiretorioProjetoLegado);
    foreach (string file in files)
    {
        FileInfo info = new FileInfo(file);

        if (info.Name.ToLower() != "program.cs" &&
            info.Extension != ".csproj" &&
            info.Extension != ".user" &&
            info.Extension != ".vspscc" &&
            info.Extension != ".sln" &&
            info.Extension != ".config")
            FileCopy(file, Path.Combine(Parametros.DiretorioProjetoConvertido, info.Name));
    }

    string[] directories = Directory.GetDirectories(Parametros.DiretorioProjetoLegado);
    foreach (string dir in directories)
    {
        DirectoryInfo infoDir = new DirectoryInfo(dir);

        if (infoDir.Name != "bin" && infoDir.Name != "obj" && infoDir.Name != "Properties")
            DirectoryCopy(dir, Path.Combine(Parametros.DiretorioProjetoConvertido, infoDir.Name), true, false);
    }
}
```

Fonte: Elaborado pelo autor (2021).

3.5 LOCALIZANDO REFERÊNCIAS DO PROJETO LEGADO

A localização das referências é uma parte muito importante da conversão de um projeto, pois como já foi mencionado anteriormente (capítulo 3), podem existir referências tanto de projetos de terceiros quanto da mesma empresa.

Quando as referências são de projetos da mesma empresa e após a conversão, no momento de executar a aplicação, ocorre algum erro por incompatibilidade de tecnologias, ter acesso ao código fonte da referência pode contribuir para conversão deste projeto, visto que se pode converter primeiro a aplicação incompatível, e posteriormente, referencia-lo manualmente.

Quando a referência é de um terceiro, e ocorre um erro no momento da compilação ou execução por incompatibilidade de tecnologia, por exemplo, dificulta bastante obter êxito na conversão, pois será necessário procurar uma referência mais recente e, dependendo o quão desatualizada estava aquela referência, podem-se ter sido descontinuados alguns métodos ou parâmetros, gerando mais trabalho para o desenvolvedor.

Foi realizada uma análise do arquivo de extensão “.csproj”, responsável pelo armazenamento das referências do projeto, analisando e comparando projetos equivalentes a fim de se chegar em um padrão e ordenação das referências de um projeto, com o objetivo de

saber onde procurar dentro do arquivo, pois existem informações que acabam não sendo relevantes para uma conversão.

O arquivo “csproj” nada mais é que um arquivo XML (*Extensible Markup Language*) que tem suas configurações separadas por tags. Existem as tags de agrupamento “PropertyGroup” e “ItemGroup”, sendo que as tags “PropertyGroup” são responsáveis por armazenar as configurações do projeto mais voltados a compilação, e as tags “ItemGroup” são responsáveis por armazenar toda e qualquer referência do projeto.

Por mais que as tags “ItemGroup” estejam agrupando diversas referências, existe uma ordem neste agrupamento, sendo que no projeto analisado existem quatro tags “ItemGroup”, sendo:

- A primeira tag está agrupando referências nativas e de terceiros.

- A segunda tag está agrupando referências voltadas ao próprio projeto, como uma nova classe, um novo form.

- A terceira tag está agrupando referências voltadas às *packages,* como por exemplo, qual versão de *framework* que o projeto utiliza.

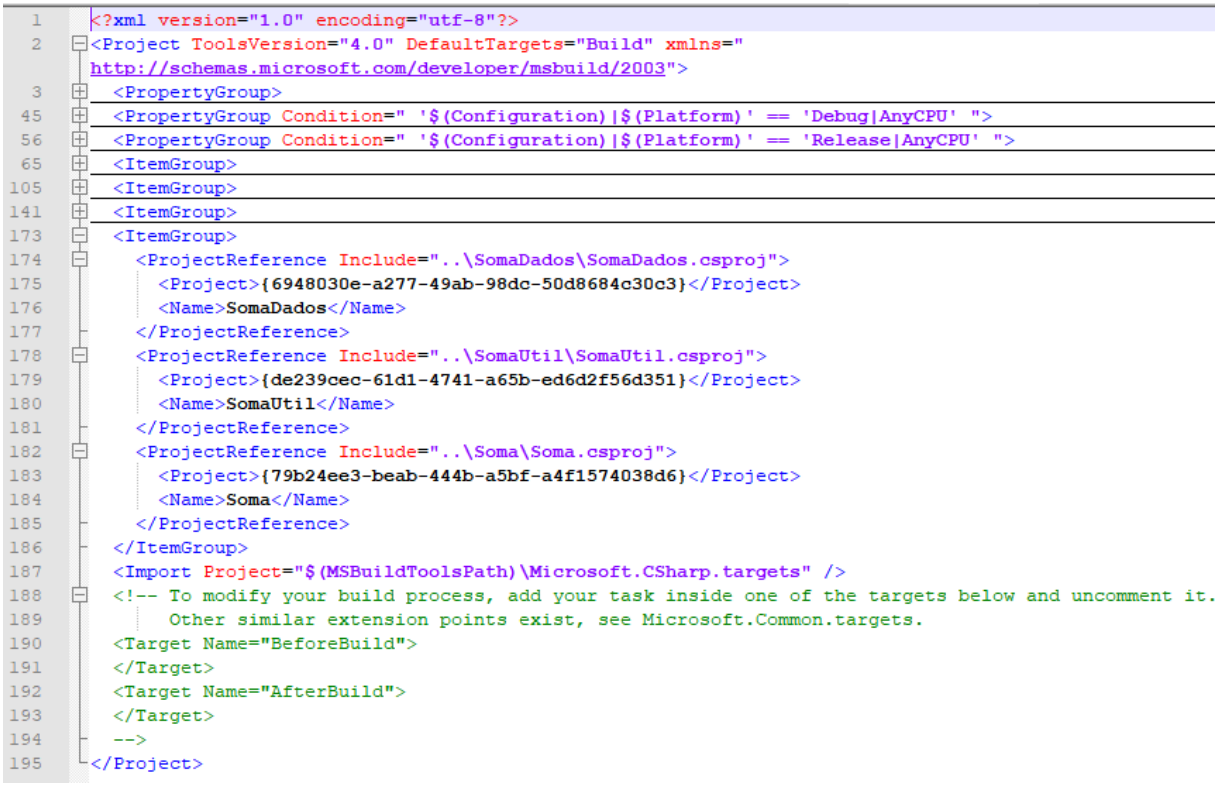
- A quarta tag está agrupando referências dos projetos locais.

Neste primeiro momento serão consideradas somente referências de projetos locais, deixando registrada a melhoria para projetos futuros.

Dentro da quarta tag, existe a tag “ProjectReference”, que é uma para cada projeto referenciado, e nela existe uma propriedade chamada “Include”, no qual está o diretório da referência. Agora dentro da tag “ProjectReference” existem duas tags, “Project” e “Name”, sendo que a informação relevante para a conversão é a tag “Name” que contém o nome da referência, já a tag “Project” contém o Id dos projetos referenciados, não sendo relevante para a conversão.

A Figura 4 ilustra o que foi descrito nesta seção.

Figura 4 - Arquivo “csproj”, responsável por guardar as referências do projeto



```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="4.0" DefaultTargets="Build" xmlns="
http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|AnyCPU' ">
  <ItemGroup>
  <ItemGroup>
  <ItemGroup>
  <ItemGroup>
    <ProjectReference Include="..\SomaDados\SomaDados.csproj">
      <Project>{6948030e-a277-49ab-98dc-50d8684c30c3}</Project>
      <Name>SomaDados</Name>
    </ProjectReference>
    <ProjectReference Include="..\SomaUtil\SomaUtil.csproj">
      <Project>{de239cec-61d1-4741-a65b-ed6d2f56d351}</Project>
      <Name>SomaUtil</Name>
    </ProjectReference>
    <ProjectReference Include="..\Soma\Soma.csproj">
      <Project>{79b24ee3-beab-444b-a5bf-a4f1574038d6}</Project>
      <Name>Soma</Name>
    </ProjectReference>
  </ItemGroup>
  <Import Project="$(MSBuildToolsPath)\Microsoft.CSharp.targets" />
  <!-- To modify your build process, add your task inside one of the targets below and uncomment it.
       Other similar extension points exist, see Microsoft.Common.targets.
  <Target Name="BeforeBuild">
  </Target>
  <Target Name="AfterBuild">
  </Target>
  -->
</Project>
```

Fonte: Elaborado pelo autor (2021).

3.6 CONVERTENDO OS FORMS

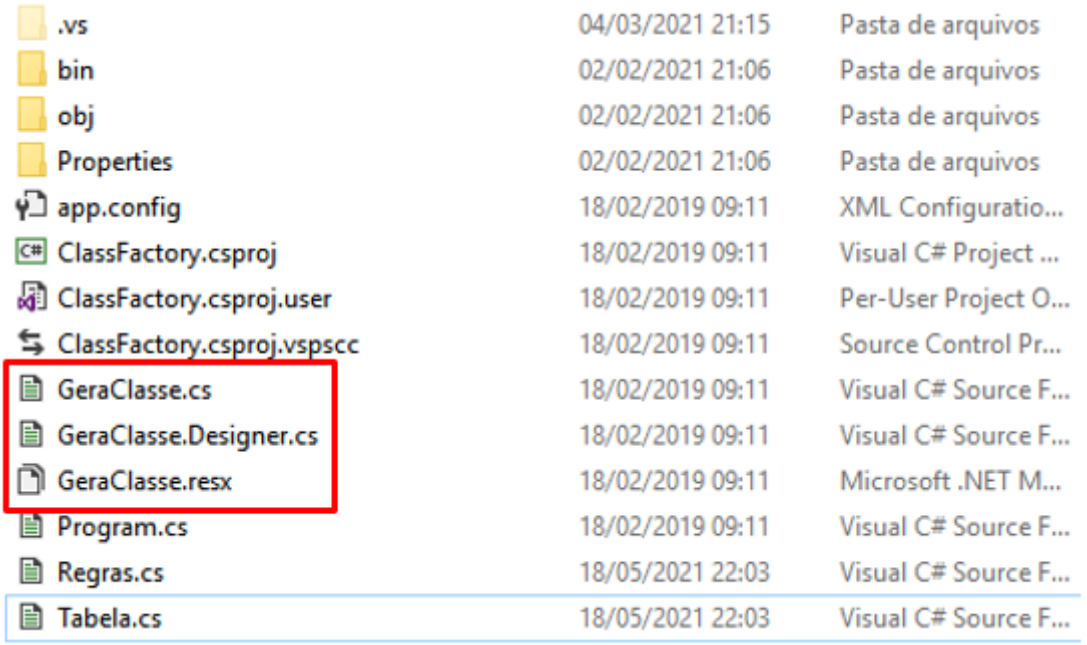
Nesta etapa é onde acontecerá a maior parte da conversão, visto que neste momento serão convertidas as telas do projeto legado para as páginas Web.

A primeira coisa a se fazer é localizar todos os forms existentes no projeto legado, inclusive os forms que estão dentro de uma hierarquia de pastas, e para isto está sendo utilizado uma classe nativa do .NET chamada DirectoryInfo que pertence a biblioteca System.IO (MICROSOFT, 2021a). Nesta classe existe uma opção de busca que permite que seja retornada toda hierarquia em um único nível, facilitando a busca. Logo após é realizada uma varredura em todos os arquivos que foram retornados, e foram desconsideradas as pastas: “bin”, “obj” e “Properties”, pois como já foi explicado anteriormente, essas pastas carregam arquivos que não possuem relevância para a conversão.

A lógica empregada nesta varredura foi a de localizar todos os arquivos que possuem a extensão “.resx” e guardá-los em uma lista, pois cada form possui três arquivos de mesmo nome mas de extensões diferentes, sendo elas: “.cs”, “.resx” e “.Designer.cs”. Os arquivos que serão analisados durante a conversão serão os com extensão “.cs”, que contém todo código

dos eventos da tela, e o arquivo de extensão ".Designer.cs" contém todos os elementos utilizados na tela, juntamente com a sua localização dentro do form, e é este arquivo que será analisado para ser gerada a página Web equivalente. A Figura 5 ilustra os arquivos que compõem o form do projeto legado.

Figura 5 - Arquivos que compõem um form



Fonte: Elaborado pelo autor (2021).

A próxima etapa consiste em efetuar uma varredura na listagem populada anteriormente, e agrupar pelo nome do arquivo. A cada iteração será aberto um arquivo ".Designer.cs" para localizar todos os componentes nele presentes. Como o arquivo possui sintaxe C#, foi realizada uma pesquisa a fim de localizar uma biblioteca que pudesse encontrar métodos, propriedades, bem como para facilitar a localização dos componentes, mas não se obteve sucesso, então foi utilizado o método força bruta (WIKIPÉDIA, 2017a), ilustrado na Figura 6, efetuando a localização do método "InitializeComponent", onde estão alocados todos os recursos utilizados no form que está sendo analisado, mas isto só foi possível porque existe um padrão de indentação dentro dos arquivos, assim sendo possível obter exatamente o método requerido, e em seguida foram listados todos os componentes e os detalhes dos componentes, bem como sua localização, seu tamanho e eventos.

Figura 6 - Método responsável por localizar as funções no código do projeto legado por meio de força bruta

```
private static string GetMetodo(string arquivo, string nomeMetodo)
{
    int index = arquivo.IndexOf(nomeMetodo);
    string aux = arquivo.Substring(index);
    index = aux.IndexOf("{\r\n");
    aux = aux.Substring(index);
    List<char> pilhaChaves = new List<char>();
    int indexMetodo = 0;
    foreach (char c in aux)
    {
        indexMetodo++;

        if (c == '{')
            pilhaChaves.Add(c);
        else if (c == '}')
            pilhaChaves.RemoveAt(pilhaChaves.Count - 1);

        if (pilhaChaves.Count == 0)
            break;
    }
    aux = aux.Substring(1, indexMetodo - 2).Trim();
    return aux;
}
```

Fonte: Elaborado pelo autor (2021).

Em uma conversão de tecnologias, quase sempre existirá algum componente que não está presente na tecnologia destino, o projeto legado utilizado como base de desenvolvimento deste *framework* possui duas, "BindingSource" e “SaveFileDialog”. No WinForms existe um componente propriamente dito, mas para Web não, o comportamento destes componentes são emulados de formas distintas.

Com base na complexidade deste problema, foi necessário desenvolver uma lógica que convertesse estes componentes em algo pré-determinado, mas de maneira dinâmica. O componente "BindingSource" no WinForms, está ligado diretamente aos componentes de tabelas (DataGridView), com base nisso, foi feita uma busca pela palavra “.BindingSource” em todos os componentes utilizados no form que está sendo convertido, a fim de saber se este form utiliza tal componente.

De posse desta informação, é marcada uma propriedade de controle dentro do código para indicar que este form que está sendo convertido faz o uso do componente “BindingSource”. Cada componente mapeado possui uma “tipagem”, dentre estas: caixas de

textos (TextBox), legendas (Label), Botões (Button). Ao todo são 66 componentes nativos que estão disponíveis para uso em uma aplicação WinForms (MICROSOFT, 2021b).

Foi realizado o mapeamento de cada componente localizado no projeto legado, e para cada componente foi criada uma conversão específica. Ao todo, foram localizados 6 componentes no form do projeto legado, e os mesmos foram convertidos para os elementos html que serão listados a seguir, sendo que estes componentes estão sendo abrangidos pelo *framework* no seu estado atual de desenvolvimento.

Componente WinForms - Elemento html;

- TextBox - input;
- Label - label;
- Button - button;
- SaveFileDialog - não foi substituído por nenhum componente, e sim foi realizado o *download* do arquivo processado;
- BindingSource - ViewBag;
- DataGridView - table, com plugin DataTables;

Foram separadas as propriedades dos componentes em duas listas relacionadas. A primeira lista guarda o tipo e o nome do componente, enquanto a segunda guarda os detalhes deste componente, sendo estes: posição, tamanho, definição de eventos, e demais propriedades. Em seguida foi percorrida esta segunda lista, e como as listas estão relacionadas, é possível saber qual o tipo de componente e nome em cada iteração.

Existem algumas propriedades de componentes que são comuns para quase todos os componentes, como: "*Location*" e "*Size*", pois se o componente está visível, ele possui uma localização e uma dimensão. Com base nisto, a primeira verificação a cada iteração são estas duas propriedades, seguida das verificações por tipo de componente. Então estas informações já são convertidas para os padrões Web, que são: "*left"* e "*top"* para "*Location*", e "*width*" e "*height*" para "*Size*".

Ainda nesta mesma iteração estão as condições para cada tipo de componente. Os componentes do tipo "TextBox", que são caixas de texto para o usuário inserir informações, foram convertidos para os elementos html "input" de tipo *"text"*. O elemento "input" possui diversos tipos de comportamento, bem como: *"text"*, *"button"*, *"hidden"*, entre outros (MDN, 2021a).

Os componentes do tipo “Label”, que são usados como legendas de outros componentes ou para dizer algo para o usuário, foram convertidos para os elementos html “label”, este componente existe uma particularidade, ele possui uma propriedade chamada “AutoSize”, que quando habilitada elimina a necessidade de se informar a propriedade “*Size*”, pois como o nome já sugere, o componente se ajusta ao tamanho do texto nele inserido.

Os componentes do tipo “DataGridView”, que são tabelas, foram convertidas para um elemento html “table”, mas este elemento está ligado a um plugin externo chamado “DataTables” (DATATABLES, 2021), este plugin tem a capacidade de alterar o comportamento da tabela, tanto visual quanto funcional, por padrão o plugin adiciona paginação, ordenação, quantidade de itens por página e um campo de pesquisa, que realiza a pesquisa em todas as colunas e em todas as páginas da tabela. O componente é ilustrado pela Figura 7.

Figura 7 - Tabela de exemplo utilizando o plugin DataTables

Show 10 entries Search:

Name	Position	Office	Age	Start date
Airi Satou	Accountant	Tokyo	33	2008/11/28
Angelica Ramos	Chief Executive Officer (CEO)	London	47	2009/10/09
Ashton Cox	Junior Technical Author	San Francisco	66	2009/01/12
Bradley Greer	Software Engineer	London	41	2012/10/13
Brenden Wagner	Software Engineer	San Francisco	28	2011/06/07
Brielle Williamson	Integration Specialist	New York	61	2012/12/02
Bruno Nash	Software Engineer	London	38	2011/05/03
Caesar Vance	Pre-Sales Support	New York	21	2011/12/12
Cara Stevens	Sales Assistant	New York	46	2011/12/06
Cedric Kelly	Senior Javascript Developer	Edinburgh	22	2012/03/29
Name	Position	Office	Age	Start date

Showing 1 to 10 of 57 entries Previous 1 2 3 4 5 6 Next

Fonte: DataTables (2021).

Além destes recursos, é possível personalizar a exibição da tabela, podendo ser alterado posteriormente com algumas linhas de código via JavaScript.

Os componentes do tipo “DataGridView” possuem suas colunas separadas em outros componentes, pois as colunas são tipadas e podem validar algum tipo de dado inserido ou até mesmo aplicar alguma formatação em campos de exibição. No momento da conversão deste componente, foi necessário criar uma condição para validar se o componente possui “*Column*” no tipo do componente, para identificar se o componente da iteração é uma coluna ou é a própria tabela.

Pela ordenação dos componentes no arquivo de extensão “Designer.cs” do form que está sendo convertido, por padrão a declaração destes componentes no arquivo segue uma ordem cronológica, listado primeiro o componente “DataGridView”, seguido de suas colunas.

Na condição criada, foi realizada a inserção do elemento “table” com toda a sua estrutura, deixando mapeado uma iteração no próprio html com o uso do Razor páginas, parte que ficará responsável por popular a tabela com os dados que virão da “Model”.

Também foi necessário criar uma condição que identificasse as colunas e acumulasse em uma variável para posteriormente ser inserido na tabela, e durante este processo, foi montada a “Model”, tipada de acordo com a tipagem da própria coluna.

Após o elemento “table”, foi necessário inserir algumas configurações no arquivo JavaScript para que o “DataTables” funcione corretamente, sendo que as referências do plugin já estão referenciadas diretamente no projeto modelo. As configurações consistem em:

- Chamar a função “DataTable” para a tabela no término do carregamento do html;
- Inserir uma função jquery no arquivo JavaScript, disponibilizada no site do próprio “DataTables”;

Como pode-se observar na Figura 8.

Figura 8 - Código responsável por efetivar o funcionamento do plugin DataTables

```
paginaJS += string.Format(@"$(""#{0}"").DataTable();" + "\r\n", nome);

paginaJSFunctions += @"$.fn.dataTable.ext.order['dom-text'] = function (settings, col) {
                            return this.api().column(col, { order: 'index' }).nodes().map(function (td, i) {
                                return $('input', td).val();
                            });
                        };" + "\r\n";
```

Fonte: Elaborado pelo autor (2021).

Os componentes do tipo “SaveFileDialog” foram convertidos de maneira diferente, pois não há um elemento em html equivalente, então foi substituído este componente por um diretório fixo dentro do projeto, e qualquer arquivo gerado pelo projeto será salvo neste diretório, pois ao término da execução será feito o *download* do arquivo para o usuário.

Os componentes do tipo “Button” foram convertidos para os elementos html “button”.

Após a conversão dos componentes é verificada uma outra condição dentro da lista detalhada dos componentes, se existe algum evento. Um evento é um procedimento que determina qual ação será executada em um determinado componente (MICROSOFT, 2017).

A existência de eventos em algum dos tipos de componentes citados acima, gera a necessidade de se inserir trechos de código em outros arquivos para que se obtenha êxito na execução do evento. Como a ideia principal é construir uma aplicação dinâmica, com o intuito de se abranger o maior número de projetos compatíveis com este *framework*, não há como saber qual a ação que um determinado evento irá efetuar, ele pode fazer qualquer ação.

Com base nisto, serão criados métodos na *Controller,* dinamicamente, para cada evento mapeado. Os gatilhos destes eventos serão criados via jquery, utilizando uma requisição ajax para chamar o método criado na *Controller.*

Figura 9 - Código dinâmico responsável pela criação da controller e gatilho para eventos

```
if (eventos.Exists(a => a.Contains(".Leave")))
{
    string leave = GetMetodo(cs, nome + "_Leave");
    leave = VerificaBindingSource(leave, utilizaBindingSource, nome);
    leave = leave.Replace(nome + ".Text", "p_" + nome);
    leave = leave.Replace("return;\r\n", "return new JsonResult(\"\");");

    paginaController += string.Format(@"public IActionResult {0}_Leave(string p_{0})
                                        {{
                                            {1}
                                            return new JsonResult("""");
                                        }}" + "\r\n", nome, leave);

    paginaJS += string.Format(@"$(""#{0}"").on(""blur"", function() {{
                                    if ($(""#{0}"").val() !== """") {{
                                        var params = {{
                                            p_{0}: $(this).val(),
                                        }};
                                        $.ajax({{{
                                            type: 'POST',
                                            url: '@Url.Action(""{0}_Leave"", ""{1}"")',
                                            contentType: 'application/x-www-form-urlencoded',
                                            data: params,
                                            success: function(result) {{
                                                if (result !== ""null"")
                                                {{
                                                    alert(result.responseText);
                                                }}
                                            }},
                                            error: function(result) {{
                                                alert(""Ocorreu um erro!\n\n"" + result.responseText);
                                            }}
                                        }});
                                    }}
                                }});" + "\r\n", nome, item.Key);
}
```

Fonte: Elaborado pelo autor (2021).

Antes de inserir todo este html gerado através da conversão, foi verificado o consumo de algumas bibliotecas nativas do ASP.NET Core 5.0, e foram importadas na *Controller*, para minimizar a interação do desenvolvedor com o projeto convertido, com o propósito de resolver possíveis erros.

Como já mencionado no capítulo 3, esta tecnologia exige que sejam inseridos trechos de código em diversos arquivos para que se obtenha o funcionamento adequado, e para seguir as boas práticas. Contudo, não foi possível descobrir de maneira dinâmica o comportamento de um trecho de código de um evento, pois este código pode executar qualquer ação, desde a escrita em algum componente presente no próprio form, quanto interagir diretamente com o usuário do sistema por meio de caixas de diálogo, e mapear todas estas ações demandaria significativo esforço de desenvolvimento.

As boas práticas instruem o desenvolvimento em uma estrutura hierárquica de arquivos, sendo:

- *Controller*;
- *Interface*;
- *Service*;

O código do form atual que está sendo convertido, deveria ser inserido em uma *Service,* e as suas declarações de funções deveriam estar listadas na *Interface,* e a *Interface* deveria estar referenciada na *Controller.* Mas, como citado acima, isto demandaria um significativo esforço de desenvolvimento para construir esse *framework,* dificultando a conclusão em tempo hábil, então o código do form foi inserido diretamente dentro do método criado na *Controller*. Mesmo assim, foram criadas a *Interface* e a *Service*, deixando-as prontas para receberem funções.

Como já apontado, o código pode interagir diretamente com um componente presente no form, mas no MVC isto não é possível, pois a *Controller* não tem acesso direto aos elementos html, a *Controller* precisa passar as informações através de uma "*ViewModel*" fortemente tipada, ou tem a opção de passar através da "*ViewBag"* e "*ViewData*" fracamente tipada (MICROSOFT, 2019b).

Esta característica aumenta severamente a complexidade da conversão, então foi necessário implementar uma lógica para contornar essa situação. Foi criada uma variável do tipo lista de texto, e a cada iteração na lista de componentes, se adiciona o nome do componente nessa nova lista criada. Após o término da iteração, é feita uma iteração na lista

com os nomes dos componentes, e é realizada uma verificação se existe algum componente com a propriedade "*.Text*". Se existe, é realizada a substituição do nome deste componente para uma "*ViewData"*, deste modo a *controller* não tentará acessar um componente presente no html, diretamente.

Após a realização dos procedimentos, todos estes elementos html que estavam alocados em variáveis, são unificados, gerando o código final que será gravado no projeto modelo que foi previamente copiado para o destino escolhido pelo desenvolvedor, no início da conversão.

Esta unificação se dá por meio de substituições, textos com caracteres especiais que são únicos dentro do html, sendo possível a substituição deste texto por todos os elementos html que foram montados no decorrer da conversão.

3.7 CONTRIBUIÇÃO COM A COMUNIDADE

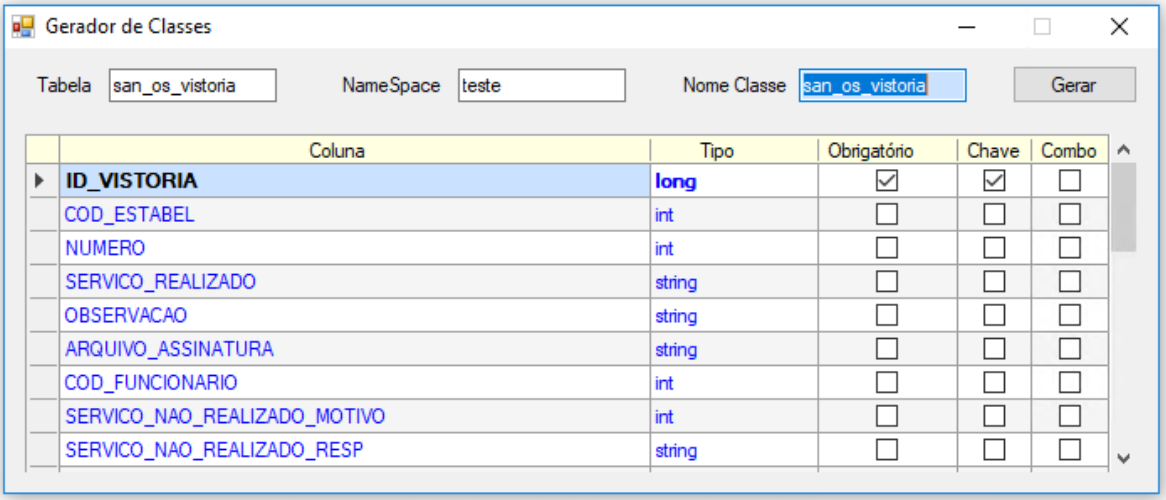
Em virtude da dedicação aqui empregada, e pelo fato de se acreditar que este trabalho agrega algum valor no segmento, para retribuir o conhecimento adquirido no desenvolvimento deste e também com o objetivo de contribuir com a comunidade, foi disponibilizado no github (https://github.com/RenanPospichil/Framework_Aplicacao) este *framework,* para que possa ser utilizado como fonte de pesquisa ou de trabalhos futuros.

É importante frisar que o código disponibilizado por este trabalho encontra-se em fase experimental, tratando-se de um protótipo e por isso mantém seu viés experimental, possivelmente não atenderá a todos os sistemas postos a prova por ele, sendo necessário intervenções e aprimoramentos por parte de quem for utilizá-lo.

3.8 CONFRONTANDO OS SISTEMAS

Nesta seção serão ilustradas as duas aplicações, a fim de efetuar uma discussão sobre os componentes utilizados e a evolução da interface gráfica. A Figura 10 ilustra o sistema original, já a Figura 11 ilustra o sistema convertido, já com os novos componentes e nova interface gráfica.

Figura 10 - Aparência da tela principal do sistema original



Fonte: Elaborado pelo autor (2021).

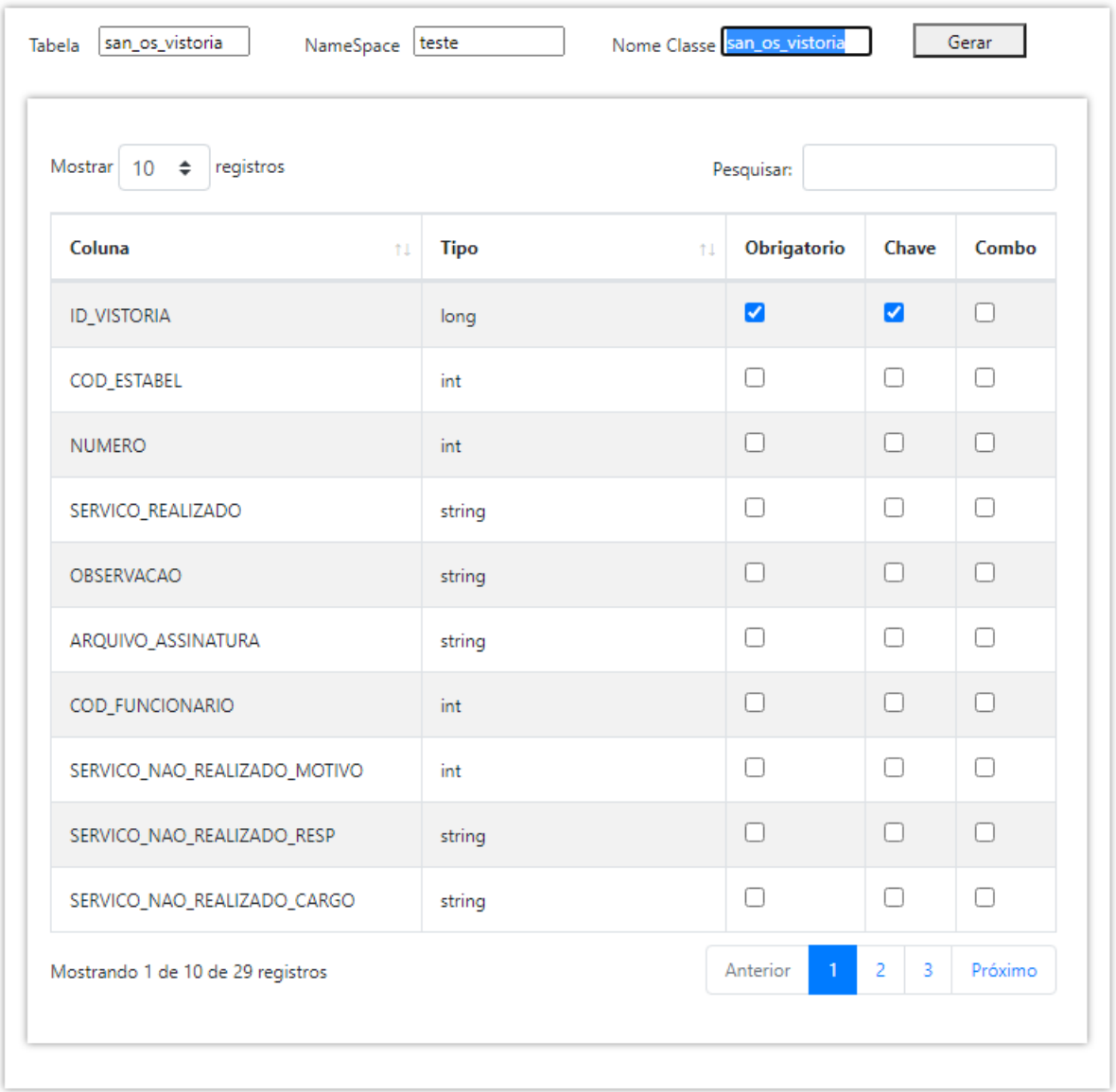
Pode-se observar na Figura 10 que este sistema é um gerador de classes, e visualmente é bastante simples, fazendo o uso de apenas 6 componentes nativos. No topo da tela, há 3 campos textos, onde o campo texto com nome “Tabela” é o campo principal, pois é ele quem dispara o evento responsável por carregar o restante da tela, bem como a tabela na parte inferior.

O funcionamento é bastante simples, o usuário informa uma tabela do banco de dados, e todas as colunas desta tabela serão carregadas na tabela presente na tela, o usuário deve informar o “NameSpace” que a classe fará parte, marcar ou desmarcar as informações pertinentes na tabela, como por exemplo, se o campo é obrigatório, se ele é chave ou até mesmo se ele terá um método que alimentará um componente comboBox.

Já o segundo campo, cujo tem bastante relevância, é o botão “Gerar”, que fica responsável por processar todas as informações e configurações informadas pelo usuário, pois o usuário pode solicitar que a classe gerada contenha mais, ou menos informações, conforme a sua necessidade. Ao pressionar o botão, será aberta uma caixa de diálogo, pedindo para informar um local para salvar o arquivo processado.

Quanto à interface gráfica, pode-se observar que todos os componentes remetem a gráficos antigos, pois se trata de uma aplicação antiga.

Figura 11 - Aparência do sistema após a conversão



Fonte: Elaborado pelo autor (2021).

O sistema convertido continua com o mesmo objetivo, ser um gerador de classes. Seu comportamento é bastante semelhante ao sistema original, com a ressalva de não abrir uma caixa de diálogo para salvar o arquivo processado. Como esta versão é Web, é simplesmente efetuado o *download* do arquivo. Contudo, o sistema convertido traz algumas melhorias na parte da tabela, adicionando recursos como: ordenação, pesquisa, paginação e quantidade de registros por página, como já citado na seção 3.7.

Quanto à interface gráfica, pode-se notar uma melhora significativa no quesito aparência, pois se utiliza elementos atuais e que podem ser facilmente customizados, sem exigir um significativo tempo de desenvolvimento.

4 EXPERIMENTOS

Neste capítulo será explicado como foram realizados os experimentos da conversão do projeto legado.

Após a conclusão do desenvolvimento do *framework* responsável por converter o projeto WinForms em Web, foi dado início a primeira etapa dos experimentos, que foram realizados durante o ato de desenvolvimento, pois foram inúmeras vezes que foram necessárias executar o *framework* para converter a aplicação e testá-la parte a parte, para averiguar se não existia erros durante a sua execução ou compilação, e a cada erro encontrado foi necessário ajustar no *framework* e repetir todo o processo novamente.

Concluindo esta primeira etapa, foi realizada a configuração de um ambiente de homologação, com o intuito de disponibilizar a aplicação original e a convertida para testes aos usuários, que por sua vez são desenvolvedores, pois o projeto legado escolhido como base de desenvolvimento, é uma aplicação de uso interno da empresa, uma ferramenta que auxilia os desenvolvedores na criação das classes utilizadas em novos projetos.

Deste modo foi possibilitado aos usuários realizar uma comparação das duas aplicações ao mesmo tempo, aumentando as suas criticidades em respeito a aplicação convertida para Web.

Como a aplicação escolhida é uma ferramenta de uso interno da empresa, este questionário foi aplicado somente aos desenvolvedores da empresa.

Para mensurar a experiência obtida pelos desenvolvedores, foi criado um questionário com perguntas referentes ao uso da aplicação convertida. Para criar este questionário foi definido um conjunto de palavras chaves para embasar questionamentos, sendo elas:

- Apresentação e interfaces gráficas;
- Funcionalidade;
- Corretude dos resultados;
- Desempenho;
- Erros e falhas;
- Compatibilidade;
- Componentes utilizados;
- Melhorias;

E com base nestas palavras-chave, foram desenvolvidas 8 perguntas objetivas e subjetivas. Nas perguntas objetivas, como mensuração foi utilizado a escala Likert (WIKIPÉDIA, 2017b).

4.1 QUESTIONÁRIO

Nesta seção será demonstrado o questionário que foi aplicado a 5 desenvolvedores, que constantemente fazem o uso do projeto legado.

Foi traçado o perfil estimado dos desenvolvedores, a ponto de qualificar esta seção de experimentos. Dos 5 desenvolvedores, 4 são seniors, com pelo menos 7 anos de experiência na área de desenvolvimento em WinForms, e 1 desenvolvedor júnior, com aproximadamente 2 anos de experiência. Os 4 desenvolvedores seniors possuem mais de 6 anos de experiência com o projeto legado, sendo que o desenvolvedor júnior possui aproximadamente 2 anos de experiência com o projeto legado.

Abaixo serão explanadas as questões presentes no questionário. As perguntas foram criadas a partir de tópicos por assunto, com o objetivo de perguntar aos desenvolvedores de maneira objetiva se a aplicação convertida atendeu as expectativas.

A primeira pergunta do questionário tem como objetivo descobrir se a apresentação e a interface gráfica utilizada atenderam as expectativas, quando comparado com uma aplicação Web convencional, do qual normalmente se está acostumado a utilizar. As respostas podem ser conferidas na Figura 12.

A segunda pergunta tem o objetivo de descobrir se durante a conversão não se perdeu nenhuma funcionalidade, quando comparado com a aplicação original. As respostas podem ser conferidas na Figura 13.

Na terceira pergunta do questionário, o objetivo é descobrir se o resultado do processamento realizado pela aplicação convertida é exatamente o mesmo que o da aplicação original. As respostas podem ser conferidas na Figura 14.

A quarta pergunta tem o objetivo de identificar se o desempenho da aplicação convertida ficou melhor, igual ou pior do que a aplicação original, visto que pelo fato de se ter utilizado tecnologias atuais, a expectativa é de uma melhora no desempenho em geral. As respostas podem ser conferidas na Figura 15.

Na quinta pergunta do questionário, o objetivo era descobrir se a aplicação convertida apresentou alguma instabilidade ou erro durante o processamento das informações, enquanto os testes eram realizados pelos desenvolvedores. As respostas podem ser conferidas na Figura 16.

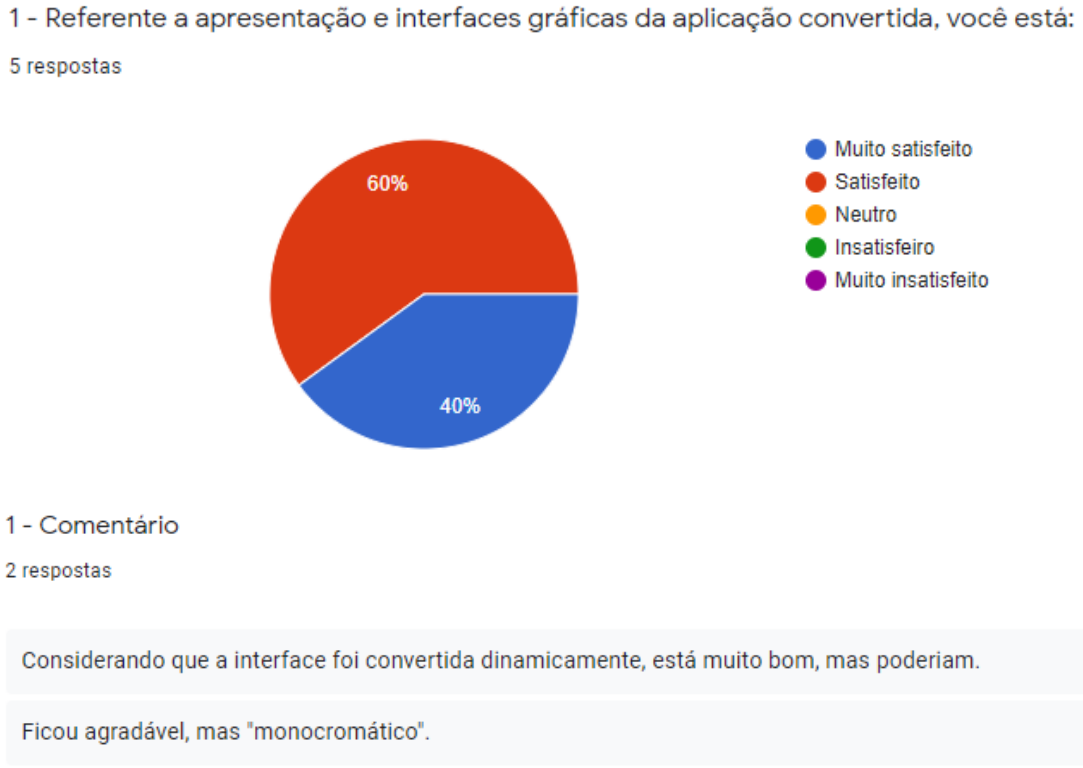
A sexta pergunta é referente a compatibilidade da aplicação convertida com os *browsers* que são normalmente utilizados, e tem o objetivo de justamente verificar se durante os testes, os desenvolvedores visualizaram alguma quebra de componente. As respostas podem ser conferidas na Figura 17.

Na sétima pergunta do questionário, o objetivo era verificar junto aos desenvolvedores, a importância do uso de um componente de Tabela, (ilustrado na Figura 7) visto que o mesmo possui alguns recursos extras, que a aplicação original não possui. As respostas podem ser conferidas na Figura 18.

E na oitava e última pergunta, o objetivo foi perguntar aos desenvolvedores se eles puderam observar algum ponto na aplicação convertida que possa ser melhorado, com isto será possível chegar em um bom resultado final. As respostas podem ser conferidas na Figura 19.

4.2 RESPOSTAS

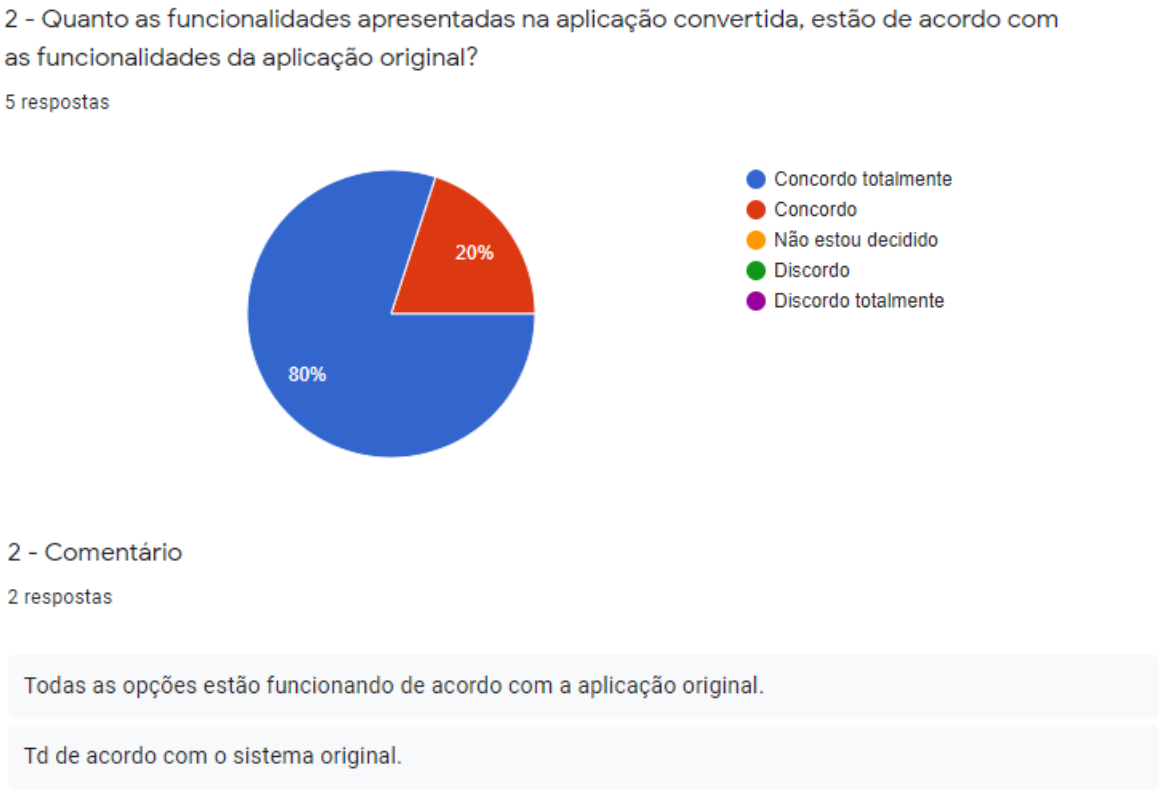
Figura 12 - Pergunta 1 do questionário



Fonte: Elaborado pelo autor (2021).

Na Figura 12 está a primeira pergunta aos desenvolvedores que executaram as duas aplicações a fim de teste e comparação, e pode-se observar que 40% dos desenvolvedores ficaram muito satisfeitos em relação a apresentação e a interface gráfica, e 60% ficaram satisfeitos. Na sessão de comentários, pode-se observar que houveram 2 comentários positivos, mas o desenvolvedor achou a aplicação convertida com poucas cores.

Figura 13 - Pergunta 2 do questionário



Fonte: Elaborado pelo autor (2021).

Na Figura 13 está a segunda pergunta do questionário, e pode-se observar que 80% dos desenvolvedores concordam totalmente que a aplicação convertida está de acordo com as funcionalidades da aplicação original, sendo que somente 20% deles, apenas concordam.

Na sessão dos comentários, houveram apenas dois comentários, sendo que os dois foram para dizer que a aplicação convertida está tudo de acordo com a aplicação original.

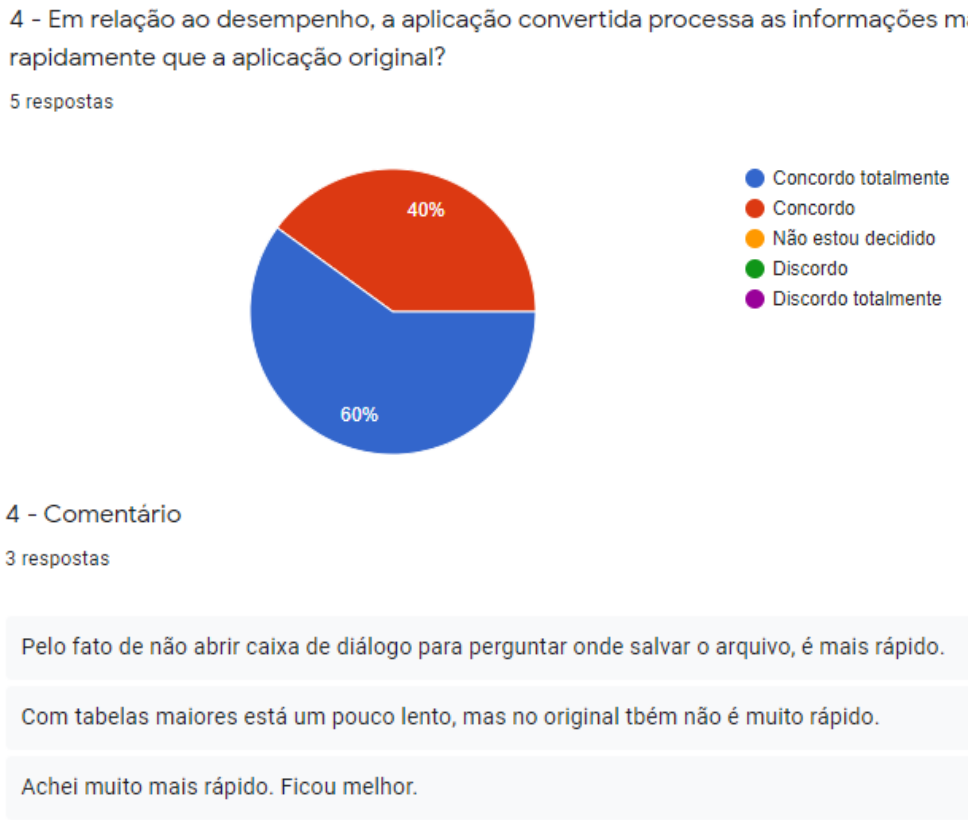
Figura 14 - Pergunta 3 do questionário

Fonte: Elaborado pelo autor (2021).

Na Figura 14 está a terceira pergunta do questionário, e pode-se observar que 80% dos desenvolvedores concordam totalmente que os resultados estão fidedignos à aplicação original, sendo que apenas 20% apenas concordam.

Nesta pergunta não houveram comentários.

Figura 15 - Pergunta 4 do questionário

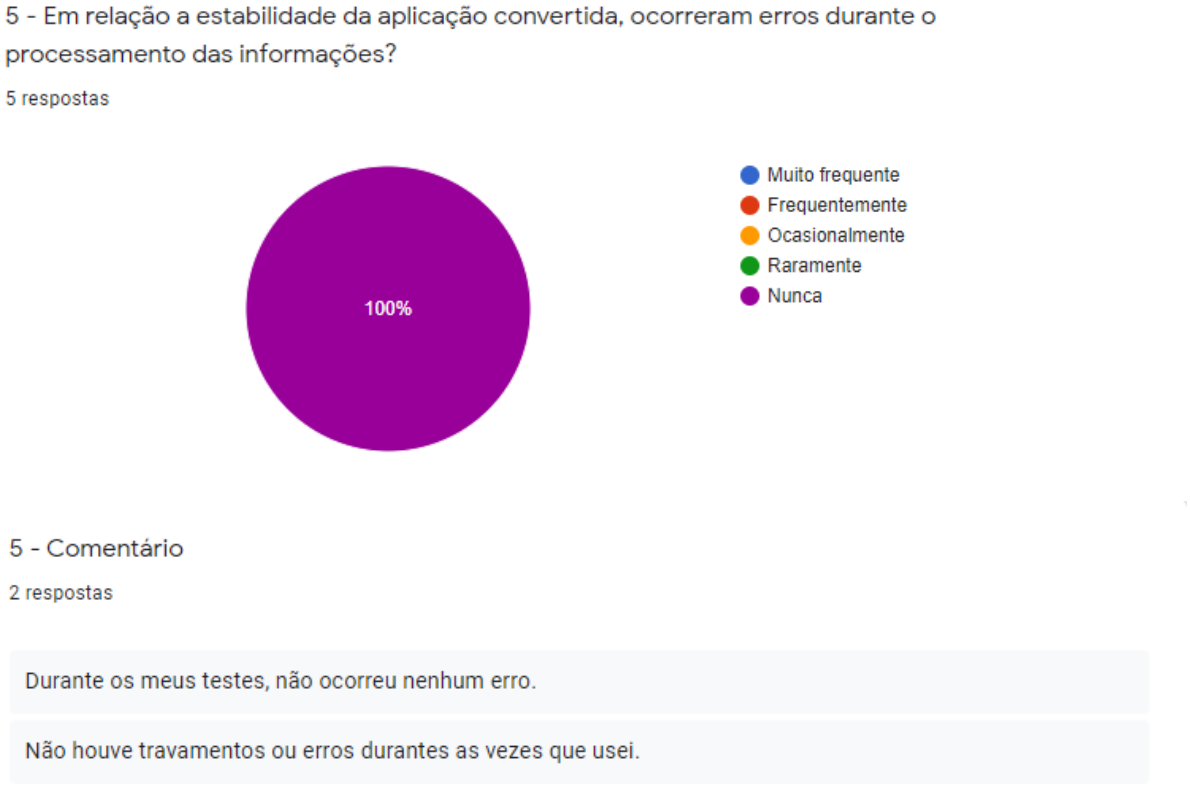


Fonte: Elaborado pelo autor (2021).

Na Figura 15 está a quarta pergunta do questionário, e pode-se observar que 60% dos desenvolvedores concordam totalmente que o desempenho da aplicação convertida processa as informações mais rapidamente que a aplicação original, sendo que 40% apenas concordam.

Na sessão dos comentários, dois desenvolvedores relataram que acharam o desempenho mais rápido na aplicação convertida do que na aplicação original, e um relatou que com tabelas grandes, demorava um pouco para processar, sendo que na aplicação original também demorava.

Figura 16 - Pergunta 5 do questionário

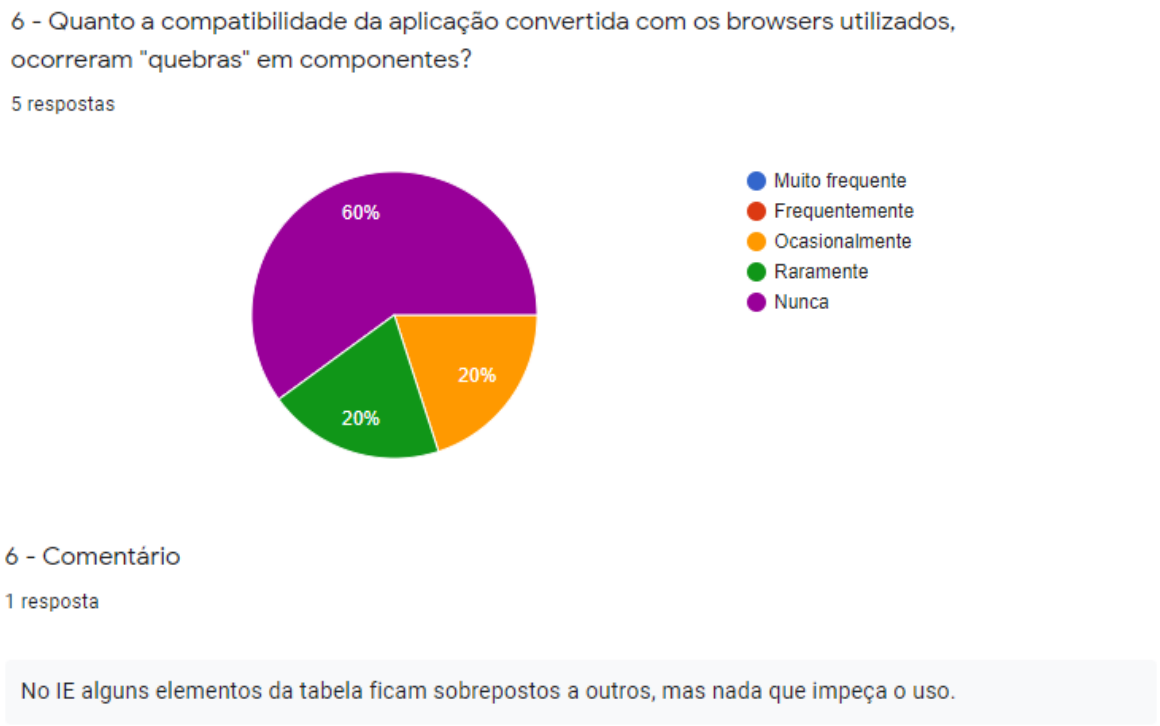


Fonte: Elaborado pelo autor (2021).

Na Figura 16 está a quinta pergunta do questionário, e pode-se observar que 100% dos desenvolvedores relataram que em nenhum momento ocorreram erros no processamento das informações.

Na sessão dos comentários, dois desenvolvedores relataram que durante os experimentos realizados, não ocorreu nenhum tipo de erro.

Figura 17 - Pergunta 6 do questionário



Fonte: Elaborado pelo autor (2021).

Na Figura 17 está a sexta pergunta do questionário, e pode-se observar que nesta pergunta houve maior divergência de opiniões, pois 60% dos desenvolvedores relataram que nunca houveram quebra em componentes durante os testes, 20% dos desenvolvedores relataram que raramente ocorrem quebras nos componentes, e 20% relataram que ocasionalmente ocorrem quebras nos componentes quando comparados em *browsers*.

Na sessão dos comentários, houve apenas um comentário dizendo que quando testado no *browser* Internet Explorer, ocorreram quebras no componente DataTable, mas nada que inviabiliza o uso.

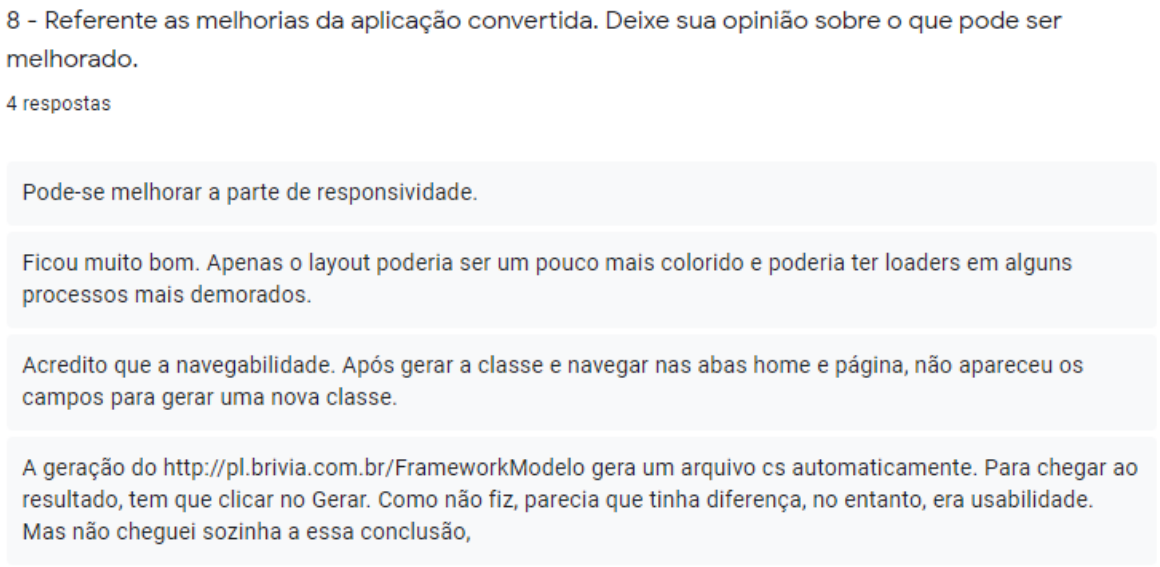
Figura 18 - Pergunta 7 do questionário

Fonte: Elaborado pelo autor (2021).

Na Figura 18 está a sétima pergunta do questionário, e nesta pergunta assim como na sexta, houveram bastante divergência de opiniões, pois 20% dos desenvolvedores julgam muito importantes os recursos presentes no componente DataTable, sendo que 60% dos desenvolvedores julgaram apenas importante, e 20% julgam moderado. Os recursos citados nesta pergunta, foram explicados no capítulo 3.6, e podem ser conferidos na Figura 7.

Na sessão dos comentários, apenas um desenvolvedor efetuou um comentário, dizendo que os recursos presentes são sempre bem vindos, mas se tratando do uso real que a tabela terá, ele julga não tão importante.

Figura 19 - Pergunta 8 do questionário



Fonte: Elaborado pelo autor (2021).

Na Figura 19 está a oitava e última pergunta do questionário, onde nesta pergunta não se foi utilizada a escala Likert, pois era esperado dos desenvolvedores algumas sugestões de melhorias para que se possa melhorar o *framework* em trabalhos futuros, e houveram quatro respostas.

A primeira sugestão comenta que se pode melhorar a responsividade. Esta questão é bastante complexa, pois boa parte do comportamento é herdado da aplicação original, mas sim, para trabalhos futuros será estudada uma maneira de melhorar a responsividade no momento da conversão.

A segunda sugestão comenta que poderia ser inserido mais cores no *design* e ser acrescentado *loaders* em processos mais demorados. Sim, poderá ser acrescentada mais cores para trabalhos futuros, talvez utilizar algum tema nativo do próprio componente, e quanto aos *loaders*, também serão acrescentados, mas provavelmente serão acrescentados em todos eventos, pois de maneira dinâmica, não há como saber se o método executa um processo demorado ou não.

Na terceira sugestão, o desenvolvedor sugeriu melhorar a navegabilidade da aplicação convertida, pois cita que após realizar o procedimento, o mesmo acessou outros menus da aplicação e não conseguiu retornar ao ponto de origem. Como esta aplicação convertida é um protótipo, existem muitas coisas que podem ser melhoradas, e pelo fato dela ser um protótipo

a navegabilidade dela talvez tenha ficado comprometida, pois na aplicação original não existe uma barra superior de menus, e na convertida existe, pois a mesma foi construída na aplicação modelo, assim, padronizando as aplicações que serão convertidas, mas ainda faltam alguns ajustes. Para trabalhos futuros este ponto também será ajustado.

Na quarta e última sugestão, o desenvolvedor comenta que encontrou dificuldades em entender o funcionamento. Bom, este ponto infelizmente não há como ser melhorado na aplicação convertida, ele deve ser melhorado na aplicação original, pois o *framework* converte os componentes antigos, em componentes novos, mas o código da aplicação original não sofre grandes modificações, apenas algumas substituições referente a algum componente, mas a lógica do código permanece a mesma, se na aplicação original não existem validações quanto ao uso, de maneira dinâmica isto não será possível implementar.

4.3 DISCUSSÃO

Nesta seção serão discutidas as respostas apresentadas pelos desenvolvedores em cada pergunta do questionário.

Considerando as respostas e comentários da primeira pergunta, pode-se observar que os desenvolvedores ficaram satisfeitos com o resultado da aplicação convertida, pois as respostas ficaram divididas entre “muito satisfeito” e “satisfeito”.

As respostas da segunda pergunta também foram muito positivas, dividindo opiniões entre “concordo totalmente” e “concordo”. Segundo os dois desenvolvedores que realizaram comentários, a aplicação está de acordo com a aplicação original.

A terceira pergunta do questionário trouxe respostas bastante satisfatórias, igualmente à segunda pergunta, dividindo as opiniões dos desenvolvedores entre “concordo totalmente” e “concordo”. É importante observar que, por mais que esta pergunta pareça semelhante à segunda, ela está questionando os resultados processados pela aplicação, e não de seus campos de interação, como questionado na segunda pergunta.

Na quarta pergunta, as respostas obtidas também foram satisfatórias já que ficaram nas mesmas estatísticas das perguntas anteriores. Esta pergunta é muito interessante e importante, pois está questionando o desempenho da aplicação convertida e, conforme os relatos dos desenvolvedores, ela ficou mais rápida.

A quinta pergunta do questionário foi a primeira pergunta a conceber unanimidade na resposta. Nesta questão, todos os desenvolvedores responderam que, durante os testes, nunca obtiveram erros no processamento das informações, consumando resultados bastantes positivos para a aplicação convertida.

Já a sexta pergunta obteve a maior divergência de opiniões dos desenvolvedores, já que trouxe como resultado respostas como “nunca”, “raramente” e “ocasionalmente” para a pergunta de compatibilidade dos *browsers* e quebra de componentes html. Por mais que 60% das respostas foram em “nunca”, 40% ficaram fora do ideal, que deveriam ser todas em “nunca”. Esta questão será aprimorada para que em trabalhos futuros ela possa atender todos os *browsers* sem quebras.

A sétima pergunta do questionário também obteve bastante divergência de opiniões dos desenvolvedores, contabilizando respostas como “muito importante”, “importante” e “moderado”, com 60% das respostas em “importante”. Por mais que houveram respostas pulverizadas, esta pergunta acaba não tendo um peso tão significativo na qualidade da aplicação convertida, pois ela está atrelada a um componente Tabela, responsável pela visualização das informações que serão processadas, e essa pergunta acaba sendo em caráter de melhoria em trabalhos futuros.

E a oitava pergunta presente no questionário, foi em caráter de melhorias da aplicação convertida. Foi solicitado aos desenvolvedores para deixarem suas opiniões sobre o que poderia ser melhorado na aplicação, resultando em quatro respostas. Pode-se observar nas respostas que não houveram sugestões semelhantes, já que cada resposta aponta para alguma deficiência que foi localizada na aplicação convertida, sendo que algumas deficiências foram herdadas no momento da conversão, não sendo possível a resolução da mesma na aplicação convertida de maneira dinâmica, para isto, a mesma deve ser ajustada na aplicação original e convertida novamente.

5 CONCLUSÃO

A computação em nuvem é uma tendência recente, mas que já está bastante desenvolvida na área, pois ela traz consigo um considerável poder de processamento aliada a um baixo custo, quando comparado ao investimento necessário para se ter este mesmo equipamento nas dependências da empresa. Também traz elasticidade, pois quando for necessário maior poder de processamento, é facilmente resolvido apenas alocando mais recursos, não sendo necessária a compra de novos equipamentos.

Geralmente aliados aos servidores locais das empresas estão os sistemas. Para mudar um servidor local para a nuvem, na grande maioria dos casos, seria necessário implementar mudanças e ajustes nos sistemas. Desta maneira, mesmo acompanhando a tendência de atualizações e modernização dos sistemas, a substituição destes certamente causaria um alto transtorno, já que esta ação reflete diretamente no uso do sistema pela empresa. Todo este cenário pode resultar em atrasos na produção por causa dos treinamentos aos usuários e possivelmente causando perda de possível lucro de capital.

Neste cenário problemático de migração, caso fosse possível converter um sistema legado de uma empresa para ambiente Web, seguindo padrões atuais, seria possível migrar o sistema para a nuvem junto com o servidor, ampliando seu poder de disponibilidade já que seria possível acessar o sistema através de qualquer dispositivo. A partir desta migração haveria uma melhora na aparência gráfica, tornando o sistema mais agradável, e ainda seria o mesmo sistema, mantendo as mesmas funcionalidades, sem prejuízo no quesito aprendizado já que todos os usuários estariam habituados com ele.

Este trabalho teve como objetivo desenvolver um *framework* capaz de converter uma aplicação WinForms para a ASP.NET Core 5.0 de maneira dinâmica, utilizando as ferramentas e recursos atuais para tornar isto possível.

Os estudos realizados foram referentes a área de conversão de sistemas para a web, mas por não localizar trabalhos relacionados ao tema de forma objetiva, o assunto foi posto de forma macro para computação em nuvem. Com base nisto, foram filtrados e estudados os trabalhos relacionados com a computação em nuvem.

Considerando o nível de complexidade do problema, foi definido um sistema legado específico, a fim de montar um modelo de desenvolvimento ajustado a este sistema. Embora

seja voltado a um sistema legado específico, o *framework* foi construído de forma dinâmica, buscando abranger um maior número de sistemas e com isso aumentar a sua compatibilidade.

O desenvolvimento ocorreu com base nas pesquisas realizadas e tecnologias disponíveis no momento, visando a construção de um *framework* capaz de converter um sistema legado em um sistema atual e tecnológico, tornando possível usufruir de um *design* moderno e de recursos atuais.

Para a continuação deste trabalho, serão implementadas todas as sugestões de melhorias que foram obtidas via questionário, bem como ajustes relacionados a defeitos mais críticos, como por exemplo, a quebra dos componentes html quando utilizada a aplicação em *browsers* variados. Também será dado continuidade a modelagem de novos sistemas, novos componentes, a fim de aumentar ainda mais a compatibilidade do *framework* com outros sistemas legados.

Além disso, para retribuir o conhecimento adquirido no desenvolvimento deste trabalho e também com o objetivo de contribuir com a comunidade, este trabalho foi disponibilizado no GitHub. Lembrando que este trabalho tem viés experimental e ainda é um protótipo, possivelmente não atenderá a todos os sistemas postos a prova por ele, sendo necessário intervenções e aprimoramentos.

Todo este trabalho teve como objetivo tornar menos complexa, e possível, a migração de um sistema WinForms C# para a ASP.NET Core 5.0, minimizando as partes que seriam prejudicadas, maximizando a produtividade e seguindo a tendência da evolução dos sistemas em nuvem.

Como citado acima, para trabalhos futuros serão necessários executar alguns ajustes, dentre eles:

I. ajuste crítico, quebra de alguns dos componentes html em *browsers* distintos;
II. melhorar responsividade das aplicações convertidas;
III. melhorar tema de cores, para sair do “monocromático”;
IV. melhorar experiência, inserindo *loaders* nos eventos.

6 REFERÊNCIAS BIBLIOGRÁFICAS

ALVES, Deriks. COSTA, Marcelo. FURTADO, Maria. et al. Computação em Nuvem: Um estudo sobre seus conceitos, tecnologia e aplicação. 2017. ISSN 2237-5252. Disponível em: <http://revistapensar.com.br/tecnologia/pasta_upload/artigos/a57.pdf>. Acessado em: 01 set. 2020.

BENZ, K. (2014). Nagios/Ceilometer Integration: New Plugin Available. 2014. Disponível em: <https://blog.zhaw.ch/icclab/nagios-ceilometer-integration-new-plugin-available/>. Acessado em: 01 set. 2020.

BOOTSTRAP. Introduction. 2021. Disponível em: <https://getbootstrap.com/docs/5.0/getting-started/introduction/>. Acessado em: 17 mar. 2021.

DATATABLES. Manual. 2021. Disponível em: <https://datatables.net/manual/>. Acessado em: 17 mar. 2021.

DOCKER. Orientation and setup. 2021. Disponível em: <https://docs.docker.com/get-started/>. Acessado em: 11 mai. 2021.

FOWLER, Martin. LEWIS, James. Microservices, 2014. Disponível em: <http://martinfowler.com/articles/microservices. html>. Acessado em: 10 set. 2020.

GREENBERG, A. HAMILTON, J. R. JAIN, N. et al. VL2: A scalable and flexible data center network. ACM SIGCOMM Computer Communication Review, 2009. Disponível em: <https://dl.acm.org/doi/10.1145/1594977.1592576>. Acessado em: 29 jun. 2021.

HARTER, T. SALMON, B. LIU, R. et al. Slacker: Fast distribution with lazy docker containers, 2016. Disponível em: <https://www.usenix.org/system/files/conference/fast16/fast16-papers-harter.pdf>. Acessado em: 29 jun. 2021.

ISARD, M. BUDIU, M. YU, Y. et al. Dryad: Distributed dataparallel programs from sequential building blocks, 2007. Disponível em: <https://dl.acm.org/doi/10.1145/1272996.1273005>. Acessado em: 29 jun. 2021.

JQUERY. Jquery.ajax. 2021. Disponível em: <https://api.jquery.com/jquery.ajax/>. Acessado em: 11 mai. 2021.

JQUERY. Jquery Inputmask. 2014. Disponível em: <https://plugins.jquery.com/jquery.inputmask/>. Acessado em: 11 mai. 2021.

KLEINROCK, L. Queueing Systems: Theory, vol. 1, 1976. Disponível em: <https://epubs.siam.org/doi/pdf/10.1137/1018095>. Acessado em: 29 jun. 2021.

LARRAANAGA, P. LOZANO, J. A. Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation, 2001. Disponível em: <https://libgen.is/book/index.php?md5=BFD5D5FECEC3AF8B0812BE88ADDDC8EF> . Acessado em: 01 jul. 2021.

MDN. Input. 2021a. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTML/Element/input>. Acessado em: 11 mai. 2021.

MDN. JavaScript. 2021b. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>. Acessado em: 13 mai. 2021.

MEZGAR, Istvan. RAUSCHECKER, Ursula. The challenge of networked enterprises for cloud computing interoperability. 2014. Disponível em: <https://sci-hub.se/10.1016/j.compind.2014.01.017>. Acessado em: 01 jul. 2021.

MICROSOFT. Caixa de Ferramentas. 2021b. Disponível em: <https://docs.microsoft.com/pt-br/visualstudio/ide/reference/toolbox?view=vs-2019>. Acessado em: 11 mai. 2021.

MICROSOFT. Criando manipuladores de eventos no Windows Forms. 2017. Disponível em:<https://docs.microsoft.com/pt-br/dotnet/desktop/winforms/creating-event-handlers-in-windows-forms?view=netframeworkdesktop-4.8>. Acessado em: 11 mai. 2021.

MICROSOFT. Gerenciar referências em um projeto. 2019a. Disponível em: <https://docs.microsoft.com/pt-br/visualstudio/ide/managing-references-in-a-project?view=vs-2019>. Acessado em: 11 mai. 2021.

MICROSOFT. Introdução às Razor páginas no ASP.NET Core. 2020c. Disponível em: <https://docs.microsoft.com/pt-br/aspnet/core/razor-pages?view=aspnetcore-5.0&tabs=visual-studio>. Acessado em: 17 mar. 2021.

MICROSOFT. Novidades do .NET 5. 2020a. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/core/dotnet-five>. Acessado em: 10 mar. 2021.

MICROSOFT. Passando dados para exibição. 2019b. Disponível em:<https://docs.microsoft.com/pt-br/aspnet/core/mvc/views/overview?view=aspnetcore-5.0>. Acessado em: 11 mai. 2021.

MICROSOFT. Práticas recomendadas de desempenho do ASP.NET Core. 2020b. Disponível em: <https://docs.microsoft.com/pt-br/aspnet/core/performance/performance-best-practices?view=aspnetcore-5.0>. Acessado em: 16 mar. 2021.

MICROSOFT. System.IO Namespace. 2021a. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/api/system.io?view=net-5.0>. Acessado em: 11 mai. 2021.

NEUMEYER, L. ROBBINS, B. NAIR, A. et al. S4: Distributed stream computing platform, 2010. Disponível em: <https://sci-hub.se/10.1109/icdmw.2010.172>. Acessado em: 01 jul. 2021.

OMARA, F. A. ARAFA, M. M. Genetic algorithms for task scheduling problem, 2009. Disponível em: <https://sci-hub.se/10.1016/j.jpdc.2009.09.009>. Acessado em: 01 jul. 2021.

PARAISO, F. HADERER, N. MERLE, P. et al. A federated multi-cloud PaaS infrastructure, 2012. Disponível em: <https://sci-hub.se/10.1109/cloud.2012.79>. Acessado em: 01 jul. 2021.

POWER, R. LI, J. Piccolo: Building fast, distributed programs with partitioned tables, 2010. Disponível em: <https://www.usenix.org/legacy/events/osdi10/tech/full_papers/Power.pdf>. Acessado em: 01 jul. 2021.

RUSCHEL, Henrique. ZANOTTO, Mariana. MOTA, Wélton. Computação em Nuvem. 2010. Disponível em: <https://www.ppgia.pucpr.br/~jamhour/RSS/TCCRSS08B/Welton%20Costa%20da%20Mota%20-%20Artigo.pdf>. Acessado em: 01 set. 2020.

SANTANA, Cleber. ANDRADE, Leandro. MELLO, Brenno. et al. Teoria e Prática de Microserviços Reativos: Um Estudo de Caso na Internet das Coisas, 2019. Disponível em: <https://www.researchgate.net/publication/337031883_Teoria_e_Pratica_de_Microservicos_Reativos_Um_Estudo_de_Caso_na_Internet_das_Coisas>. Acessado em: 01 jul. 2021.

SHEN, H. ZHANG, Y. Improved approximate detection of duplicates for data streams over sliding windows, 2008. Disponível em: <https://sci-hub.se/10.1007/s11390-008-9192-1>. Acessado em: 01 jul. 2021.

SUN, J. ZHANG, Q. TSANG, E. P. K. DE/EDA: A new evolutionary algorithm for global optimization, 2005. Disponível em: <https://sci-hub.se/10.1016/j.ins.2004.06.009>. Acessado em: 01 jul. 2021.

SYED, Jamil. GANI, Abdullah. NASARUDDIN, Fariza. et al. CloudProcMon: A Non-Intrusive Cloud Monitoring Framework, 2018. Disponível em: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8432408>. Acessado em: 01 mai. 2021.

VIGLAS, S. NAUGHTON, J. F. Rate-based query optimization for streaming information sources, 2002. Disponível em: <https://sci-hub.se/https://dl.acm.org/doi/10.1145/564691.564697>. Acessado em: 01 jul. 2021.

WIKIPÉDIA. Busca por força bruta. 2017a. Disponível em: <https://pt.wikipedia.org/wiki/Busca_por_for%C3%A7a_bruta>. Acessado em: 11 mai. 2021.

WIKIPÉDIA. Escala Likert. 2017b. Disponível em: <https://pt.wikipedia.org/wiki/Escala_Likert>. Acessado em: 28 mai. 2021.

YU, J. BUYYA, R. Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms, 2006. Disponível em: <https://downloads.hindawi.com/journals/sp/2006/271608.pdf>. Acessado em: 01 jul. 2021.

ZAHARIA, M. CHOWDHURY, M. DAS, T. et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, 2012. Disponível em: <https://www.usenix.org/system/files/conference/nsdi12/nsdi12-final138.pdf>. Acessado em: 01 jul. 2021.

ZALAZAR, Ana Sofia. GONNET, Silvio. LEONE, Horacio. Migration of Legacy Systems to Cloud Computing. 2015. Disponível em: <https://www.researchgate.net/publication/280154501_Migration_of_Legacy_Systems_to_Cloud_Computing>. Acessado em: 09 out. 2020.