Otimizações em Launcher de Jogos: Análise de um Caso Prático

Bruno Henrique Araújo da Costa

CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

João Pessoa, 2024

Bruno Henrique Araújo da Costa

Otimizações em Launcher de Jogos: Análise de um Caso Prático

Monografia apresentada ao curso Engenharia de Computação
do Centro de Informática, da Universidade Federal da Paraíba,
como requisito para a obtenção do grau de Bacharel em Engenharia de Computação

Orientador: Marcelo Iury de Sousa Oliveira

Novembro de 2024

Catalogação na publicação
Seção de Catalogação e Classificação

C837o Costa, Bruno Henrique Araujo da.
Otimizações em launcher de jogos: análise de um caso prático / Bruno Henrique Araujo da Costa. - João Pessoa, 2024.
47 f. : il.

Orientação: Marcelo Iury de Sousa Oliveira.
TCC (Graduação) - UFPB/CI.

1. Launcher. 2. CI/CD. 3. Mudança de paradigma. I. Oliveira, Marcelo Iury de Sousa. II. Título.

UFPB/CI CDU 004.4

Elaborado por Michelle de Kássia Fonseca Barbosa - CRB-738

CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

Trabalho de Conclusão de Curso de Engenharia de Computaçãointitulado ***Otimizações em Launcher de Jogos: Análise de um Caso Prático*** de autoria de Bruno Henrique Araújo da Costa, aprovada pela banca examinadora constituída pelos seguintes professores:

Prof. Dr. Marcelo Iury de Sousa Oliveira
CI/UFPB

Prof. Dr. Raoni Kulesza
CI/UFPB

Prof. Dr. Derzu Omaia
CI/UFPB

Coordenador(a) do curso de Engenharia de Computação
Josilene Aires Moreira
CI/UFPB

João Pessoa, 14 de novembro de 2024

Centro de Informática, Universidade Federal da Paraíba
Rua dos Escoteiros, Mangabeira VII, João Pessoa, Paraíba, Brasil CEP: 58058-600
Fone: +55 (83) 3216 7093 / Fax: +55 (83) 3216 7117

"Tente uma, duas, três vezes e se possível tente a quarta, a quinta e quantas vezes for necessário. Só não desista nas primeiras tentativas, a persistência é amiga da conquista. Se você quer chegar aonde a maioria não chega, faça o que a maioria não faz".

Bill Gates

AGRADECIMENTOS

Gostaria de expressar minha sincera gratidão a todos que participaram, direta ou indiretamente dessa jornada até a conclusão da graduação. Primeiramente, agradeço a todos da minha família por todo o apoio incondicional e condição que me proporcionaram durante todos esse anos. A sua preocupação e dedicação durante o processo foram inestimáveis e essenciais para a finalização desse trabalho.

Também gostaria de agradecer aos amigos e colegas que estiveram comigo durante o curso dessa graduação, me auxiliando com dúvidas, oferecendo suporte e demonstrando companheirismo nas situações adversas.

Por fim, mas não menos importante, gostaria de agradecer aos membros da banca avaliadora deste trabalho, em especial ao meu orientador, Dr. Marcelo Iury de Sousa Oliveira. Também agradeço ao meu chefe, Dr. Atila Valgueiro Malta Moreira, suas contribuições, orientações e paciência foram de grande valia para a realização desse trabalho. Minha sincera gratidão à todos vocês por me auxiliarem nesse momento tão importante da minha vida.

RESUMO

A crescente evolução tecnológica impulsiona a melhoria na qualidade do software de diversas empresas, que buscando a elevação do nível de satisfação de seus clientes, abrem espaço para a utilização de novos mecanismos e ferramentas. Assim, para possibilitar a atualização constante de suas aplicações e reduzir os custos de manutenção, a adoção de práticas como *Continuous Integration and Continuous Deployment* (CI/CD) é vital para o processo. Essa abordagem traz consigo a mudança de paradigma de produto para serviço, permitindo uma entrega de valor contínua e atentando-se à experiência de usuário. Nesse contexto, o presente trabalho tem como objetivo a atualização tecnológica de um launcher de jogo, com a utilização prática dos conceitos recém citados. Para tanto, inicialmente foi realizada uma análise estrutural da aplicação inicial, bem como das suas tecnologias prévias e as que eram necessárias para o desenvolvimento de melhorias no software. Posteriormente, iniciou-se o processo de atualização tecnológica, visando os objetivos estabelecidos. Alguns dos resultados perceptíveis foram a diminuição de problemas existentes na nova versão, além da prevista redução nos custos de manutenção e melhoria, que podem servir de inspiração para futuros projetos relacionados.

Palavras-chave: Launcher, CI/CD, Mudança de paradigma.

ABSTRACT

The growing technological evolution drives improvements in software quality across various companies, which, in seeking to enhance customer satisfaction, create opportunities for the use of new mechanisms and tools. Thus, to enable constant updates to their applications and reduce maintenance costs, the adoption of practices such as Continuous Integration and Continuous Deployment (CI/CD) is vital for the process. This approach brings with it a paradigm shift from product to service, allowing for continuous value delivery while focusing on user experience. In this context, the present work aims to technologically update a game launcher, practically utilizing the aforementioned concepts. To this end, an initial structural analysis of the application was carried out, as well as its previous technologies and those necessary for the development of improvements in the software. Subsequently, the technological update process was initiated, aiming to meet the established objectives. Some noticeable results included a reduction in existing issues in the new version, as well as the expected decrease in maintenance and improvement costs, which can inspire future related projects.

Palavras-chave: Launcher, CI/CD, Paradigm Shift.

LISTA DE FIGURAS

1 Layout do Steam em 2003. Fonte: PCGamer.com (2022) 17

2 Layout do Steam em 2006, com a presença de jogos de terceiros. Fonte: webdesignmuseum.org (2006) 18

3 Layout do Steam em 2013, com a presença da aba de softwares. Fonte: webdesignmuseum.org (2013) 19

4 Layout do Steam em 2024. Fonte: webdesignmuseum.org (2024) 20

5 Pipeline de CI/CD. Fonte: mindbowser.com (2021) 21

6 Visualização da versão inicial do *launcher*. Fonte: Raid Hut (2024) 22

7 Visualização da versão inicial do *launcher* com um servidor instalado. Fonte: Raid Hut (2024) 23

8 Visualização da barra superior do *launcher*. Fonte: Raid Hut (2024) 23

9 Trecho do arquivo JSON que é lido pelo *launcher*. Fonte: Raid Hut (2024) 24

10 Visualização do sistema de autenticação com falha. Fonte: Raid Hut (2024) 25

11 Visualização da seção lateral do *launcher*. Fonte: Raid Hut (2024) 25

12 Visualização da seção principal do *launcher*. Fonte: Raid Hut (2024) . . . 27

13 Exemplo de visualização de mapa de calor em aplicação com Clarity. Fonte: Microsoft (2024) 30

14 Trecho de código referente à configuração do Electron Forge. Fonte: Raid Hut (2024) 31

15 Trecho de código referente ao pipeline de CI/CD. Fonte: Raid Hut (2024) . 31

16 Trecho de código referente à checagem de atualizações do *launcher*. Fonte: Raid Hut (2024) 32

17 Visualização do *launcher* contendo um QR Code para login. Fonte: Raid Hut (2024) 33

18 Visualização do *launcher* com usuário autenticado. Fonte: Raid Hut (2024) 34

19 Exemplo de utilização dos métodos principais da biblioteca. Fonte: Raid Hut (2024) 35

20 Visualização do mapa de calor do *launcher* no Microsoft Clarity. Fonte: Raid Hut (2024) 35

21 Trecho de código referente ao teste da barra superior. Fonte: Raid Hut (2024) 37
22 Avaliação do tempo gasto em atividades manuais. Fonte: Autor (2024) . . 40
23 Avaliação da agilidade na integração de novas funcionalidades. Fonte: Autor (2024) 40
24 Avaliação do aumento na segurança do processo de deploy. Fonte: Autor (2024) 41
25 Avaliação do aumento na confiança da qualidade das versões. Fonte: Autor (2024) 41
26 Avaliação do aumento na colaboração entre as equipes. Fonte: Autor (2024) 42
27 Avaliação da melhora de entrega de valor. Fonte: Autor (2024) 42
28 Avaliação da agilidade na entrega de novos recursos. Fonte: Autor (2024) . 43
29 Avaliação do aumento na colaboração entre as equipes. Fonte: Autor (2024) 43
30 Avaliação da redução na quantidade de erros que chegam ao usuário. Fonte: Autor (2024) 44
31 Avaliação da constante diminuição do número de reclamações referente ao *launcher*. Fonte: Autor (2024) 44

LISTA DE TABELAS E QUADROS

1 Pontos de falha percebidos 28

2 Benefícios observados para cada *stakeholder* 38

LISTA DE ABREVIATURAS

AI *Artificial Intelligence*.

API *Application Programming Interface*.

CD *Continuous Deployment*.

CI *Continuous Integration*.

CI/CD *Continuous Integration and Continuous Deployment*.

HL2 *Half-Life 2*.

JSON *JavaScript Object Notation*.

QR Code *Quick-Response Code*.

S3 *Amazon Simple Storage Service*.

VR *Virtual Reality*.

WYD *With Your Destiny*.

Sumário

1 INTRODUÇÃO **14**

1.0.1 Objetivo geral . . . 15

1.0.2 Objetivos específicos . . . 15

1.1 Metodologia . . . 16

1.2 Estrutura do trabalho . . . 16

2 CONCEITOS GERAIS E REVISÃO DA LITERATURA **17**

2.1 Launchers . . . 17

2.2 CI/CD . . . 20

3 ANÁLISES E EXECUÇÃO **22**

3.1 Análise estrutural da arquitetura . . . 22

3.1.1 Barra superior . . . 23

3.1.2 Seção lateral . . . 25

3.1.3 Seção principal . . . 27

3.2 Análise tecnológica . . . 28

3.2.1 Tecnologias pré-existentes . . . 28

3.2.2 Atualizações de bibliotecas . . . 29

3.3 Execução do processo de atualização tecnológica . . . 30

3.3.1 Configurações iniciais . . . 30

3.3.2 Realocação dos componentes . . . 32

3.3.3 Implementação de novas funcionalidades e correção de problemas . 33

3.3.4 Implantação de testes unitários . . . 36

3.4 Resultados . . . 38

3.4.1 Resultados gerais . . . 38

3.4.2 Análise de percepção . . . 39

4 CONCLUSÕES E TRABALHOS FUTUROS **45**

REFERÊNCIAS **45**

1 INTRODUÇÃO

A indústria de games tem sido impactada diretamente pelo avanço da tecnologia. O setor atualmente gera uma receita global superior ao da indústria da música e da indústria cinematográfica somadas (Arora, 2023). Em seus primórdios, contava apenas com jogos de jogabilidade simples e limitada à um determinado dispositivo. Porém, com o uso de tecnologias mais avançadas em games tem se tornado cada vez mais frequente Carvalho et al. (2016). Ficheman et al. (2011) exemplificam esse comportamento com o jogo GP Brasil VR, que utiliza amplamente tecnologias como *Artificial Intelligence* (AI) e *Virtual Reality* (VR). Tais inovações buscam melhorar a experiência do usuário e estabelecer novos padrões de qualidade tecnológica no desenvolvimento de jogos.

Algumas empresas surgem para assumir um papel importante no mercado de jogos: ser uma *publisher* de games. Diferentemente de uma desenvolvedora, a qual tem por função realizar a criação em si do jogo, uma *publisher* é responsável pela distribuição desse produto no mercado, buscando que este tenha o maior alcance possível. Embora as *publishers* não sejam responsáveis pelo desenvolvimento do produto, é necessário realizar um conjunto de ações em torno dele a fim de melhorá-lo. A busca por maior engajamento através da elaboração de produtos secundários e a coleta de *feedbacks* dos usuários são bons exemplos de ações a serem realizadas nesse sentido.

Por muito tempo, o principal meio de distribuição de jogos era por meio do uso de mídias físicas, por exemplo CDs, DVDs. Através dessas mídias era possível realizar a completa instalação e execução de um programa. Um motivo fundamental para a utilização desses artifícios era a evidente limitação das velocidades de conexão existente no período, que inviabilizava a distribuição de grandes volumes de dados pela Internet.

No entanto, muitos problemas decorriam dessa prática, entre eles, a dificuldade do usuário em conseguir atualizações para o software quando fosse necessário. Por vezes, era necessário que o utilizador adquirisse outro disco contendo um *patch*, dessa forma, corrigindo possíveis problemas.[1] De acordo com George (2013), no contexto de evitar ter que ir à uma loja adquirir a mídia física, a pirataria se mostrava uma opção agradável aos olhos do utilizador.

Assim, surge o conceito de *launcher*, que nada mais é do que um software de gerenciamento, atuando como intermediário no relacionamento entre o usuário e o programa final. Entre as funcionalidades mais comuns dos *launchers*, pode-se destacar a detecção, download e instalação de novos *patches*, a pré-configuração do software final com as preferências do utilizador, além de permitir o acesso à conteúdos secundários. Os *launchers* passaram a ser tratados como serviços, adotando práticas como CI/CD permitindo que

[1]Software Through the Ages. Medium, 2020. Disponível em: `https://medium.com/cardstack/software-through-the-ages-7ae7b3debfd7`. Acesso em: 16/09/2024.

correções e melhorias sejam aplicadas ao software do usuário de maneira contínua e automatizada, minimizando o tempo e esforço por parte do desenvolvedor e do usuário.

Nesse contexto, a atualização contínua do *launcher* é um processo importante para garantir sua estabilidade, segurança e desempenho. Além disso, é importante minimizar o esforço necessário por parte do usuário para que tenha seu *launcher* na versão mais recente, principalmente evitando que seja preciso baixá-lo novamente. Kanerva (2024) pontua que atualizações regulares em softwares são cruciais para prevenir potenciais falhas de segurança, minimizando a possibilidade de usuários realizarem alguma ação maliciosa. Desse modo, é necessário que a migração de paradigma de produto para serviço seja realizada na aplicação, a fim de obter um maior grau de satisfação entre os usuários, tornar a aplicação mais segura e minimizar as etapas manuais durante a fase de desenvolvimento.

1.0.1 Objetivo geral

O objetivo deste relatório técnico é apresentar uma visão geral de um novo *launcher* desenvolvido pela empresa Raid Hut[2], uma publisher de jogos com sede no Brasil, especificamente na cidade de Recife - PE. Para tanto, uma abordagem de migração de paradigma de produto para serviço é utilizada. Essa implementação foi realizada especificamente para o jogo *With Your Destiny* (WYD), cuja responsabilidade de desenvolvimento é da empresa sul-coreana JoyImpact.

1.0.2 Objetivos específicos

Este trabalho possui alguns objetivos específicos para a sua execução, que são:

1. Avaliação estrutural:

   (a) Análise das estruturas da versão inicial do *launcher*, observando pontos tais como: arquitetura, escalabilidade e manutenção.

2. Avaliação das tecnologias:

   (a) Identificação de tecnologias utilizadas na versão inicial do *launcher*, comparando-as com as em prática na indústria para a arquitetura de softwares similares.

   (b) Identificação de tecnologias que necessitam ser adicionadas ao projeto, com o intuito de trazer as novas funcionalidades previstas ao *launcher*.

3. Realização do Processo de Atualização Tecnológica:

[2]About us. Raid Hut, 2023. Disponível em: `https://www.raidhut.biz/about-1`. Acesso em: 16/09/2024.

(a) Condução do plano de transição do *launcher*, seguindo um cronograma rigoroso para garantir a integridade do processo.

(b) Implementação de melhorias na arquitetura e no código, conforme eventual necessidade observada na avaliação estrutural.

(c) Condução de uma fase de testes, garantindo que o mesmo atenda aos requisitos de qualidade e performance, sem que haja impacto negativo nos usuários.

1.1 Metodologia

O principal objetivo deste trabalho é realizar o estudo técnico em um *launcher* de jogo acerca do processo de migração de paradigma de produto para serviço, através da implantação de um pipeline de CI/CD. O projeto foi constituído por uma equipe de duas pessoas, utilizando a metodologia *Scrum*[3] para a gerência e organização do trabalho. O uso dessa metodologia permitiu um cenário de melhoria contínua e maior agilidade nas entregas de valor.

1.2 Estrutura do trabalho

Este trabalho está dividido em três partes principais: conceitos gerais e revisão da literatura (capítulo 2), análises e execução (capítulo 3) e conclusões e trabalhos futuros (capítulo 4). O capítulo de conceitos gerais e revisão da literatura aborda os conceitos de *launcher* e CI/CD, sendo os principais artifícios utilizados para a execução do trabalho.

A análise e execução traz o ponto de vista da prática, mostrando como a execução foi dividida e implementada de fato, visando atingir o objetivo principal, além de possibilitar a visualização dos resultados práticos. Finalmente, o capítulo de conclusões e trabalhos futuros discorre sobre alguns pontos relevantes, tais como as dificuldades encontradas durante o trabalho, bem como as lições aprendidas e propostas para trabalhos futuros.

[3]Scrum Methodology Explained: An Introduction for Agile Teams. Nimblework, 2024. Disponível em: `https://www.nimblework.com/agile/scrum-methodology/`

2 CONCEITOS GERAIS E REVISÃO DA LITERATURA

No presente capítulo, serão apresentados dois conceitos principais para o desenvolvimento do trabalho: os *launchers* e as práticas de CI/CD. Tais conceitos são importantes para entender o objeto de estudo do trabalho, bem como a principal metodologia aplicada para a melhoria da eficiência da aplicação.

2.1 Launchers

Em meados dos anos 80, a distribuição de jogos era predominantemente realizada através de mídias físicas, visto que o acesso à internet ainda não era comum ao redor do mundo e as velocidades de conexão eram altamente limitadas. No entanto, no início dos anos 2000, com a constante evolução tecnológica e das velocidades de conexão com a internet, torna-se viável o modelo de distribuição digital de jogos, representando um marco nos métodos de distribuição, conforme apontado por Rowe (2014).

Uma das iniciativas pioneiras nesse contexto se deu por parte da Valve Corporation, que em 2003 apresentou seu *launcher*, o Steam, inicialmente voltado para ser o portal único de acesso para seu jogo mais aguardado: o *Half-Life 2* (HL2). Dessa forma, o intuito inicial da companhia era evitar ao máximo problemas com pirataria e usuários não licenciados, além de implementar o processo de entrega de melhorias e correção de problemas para o jogo automaticamente.

Figura 1: Layout do Steam em 2003. Fonte: PCGamer.com (2022)

Através da solução proposta pela empresa, a plataforma diminuiu as barreiras existentes para usuários e desenvolvedores, mudando suas respectivas formas de lidar com os jogos. Com a grande expectativa pelo lançamento do jogo por parte da comunidade, um número expressivo de cópias do HL2 foi vendido, resultando também em inúmeros casos de pirataria, onde, de acordo com Dymek (2005 apud Fahey, 2004) cerca de 20,000 contas foram suspensas na plataforma por tentativa de pirataria.

Em contraste com as vantagens oferecidas pelo modelo, houve uma sobrecarga nos servidores da companhia devido à grande quantidade de acessos simultâneos no dia do lançamento do título, fazendo com que boa parte da comunidade não conseguisse sequer iniciar o jogo. Assim, um dos primeiros problemas que seriam encontrados no caminho foi revelado, gerando insatisfação dentro da comunidade de jogadores, que passaram a questionar a validade do modelo. (Joelsson et al. 2018)

Embora a sobrecarga tenha sido resolvido rapidamente, permitindo que diversos jogadores acessassem o jogo em poucos dias, um considerável grau de desconfiança ainda permaneceu na comunidade. Com o tempo, em meados de 2006 a empresa firmou contratos com diversas *publishers*, permitindo que seus respectivos jogos estivessem à venda na plataforma, fato este que gerou uma mudança na percepção das pessoas em relação ao Steam.

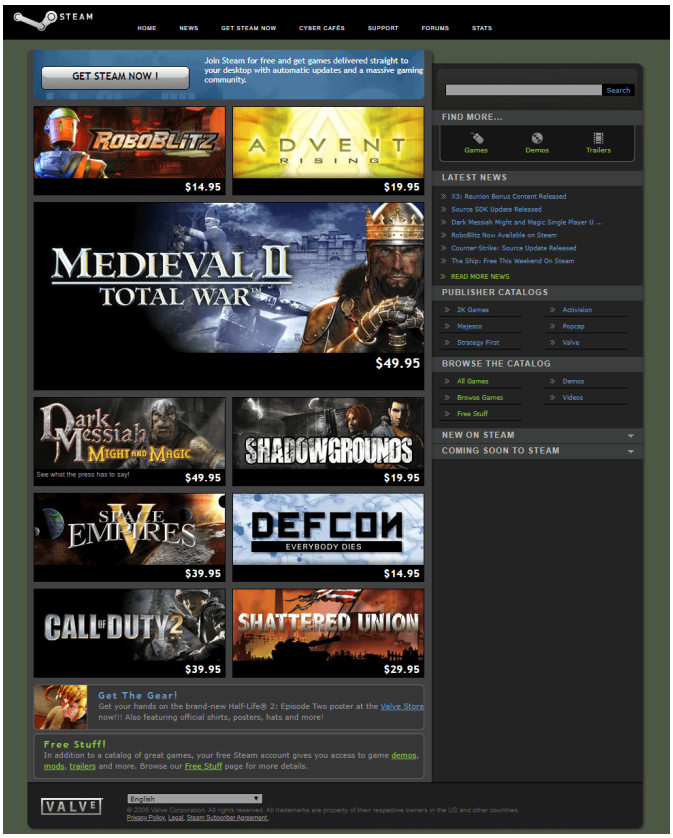

Figura 2: Layout do Steam em 2006, com a presença de jogos de terceiros. Fonte: webdesignmuseum.org (2006)

A possibilidade de terceiros publicarem jogos na Steam denotou o marco inicial na evolução da plataforma, em que, o launcher além de possuir as funcionalidades previstas, agora contava com uma loja de jogos digitais. Posteriormente, foi desenvolvida a ferramenta *SteamWorks* que trazia consigo a possibilidade de desenvolvedores publicarem seus títulos sem a necessidade da Valve intermediar o processo, acelerando o processo e tornando-o menos burocrático.

Por volta de 2012, o launcher recebeu a adição de um sistema de armazenamento de conteúdo feita por usuários, o Steam Workshops, permitindo que usuários utilizassem sua criatividade para criar conteúdos para seus jogos favoritos, podendo receber prestígio por isso. Além disso, de acordo com Joelsson et al. (2018), a Valve permitiu que softwares comuns pudessem ser adicionados à loja, finalizando uma era exclusiva para os games e dando início à um ecossistema de softwares.

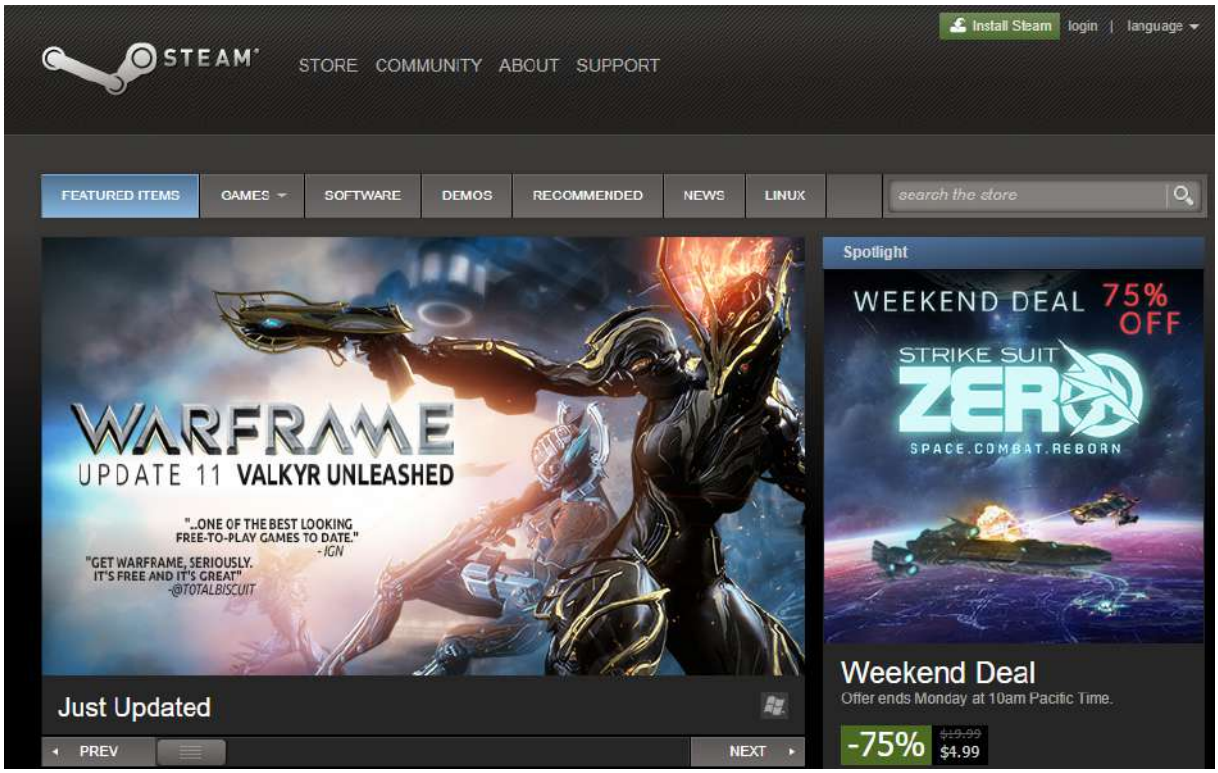

Figura 3: Layout do Steam em 2013, com a presença da aba de softwares. Fonte: webdesignmuseum.org (2013)

No final de 2015, a companhia adentrou o mundo da VR com o lançamento do *HTC Vive*, um óculos de realidade virtual desenvolvido em parceria com a empresa Vive. Além disso, foi desenvolvida a plataforma SteamVR que permite que os jogadores desfrutem dos jogos com a tecnologia VR comprados através da loja. [4]

[4]Steam: the evolution of UI and UX in Gaming. UX Collective, 2023. Disponível em: `https://uxdesign.cc/steam-the-evolution-of-interface-and-user-experience-in-gaming-42159c8b8ad1`. Acesso em: 25/09/2024.

Além disso, além da disponibilização de softwares, foram adicionados ao catálogo uma coletânea de filmes e programas de televisão, aproximando-se de um público-alvo diferente.[5] Desde então, melhorias contínuas vêm sendo aplicadas ao *launcher*, principalmente no que se refere à experiência do usuário.

Figura 4: Layout do Steam em 2024. Fonte: webdesignmuseum.org (2024)

2.2 CI/CD

No contexto de um time de desenvolvedores, o processo padrão de desenvolvimento de software por muito tempo se tratou de um processo lento e manual, onde, mudanças incorporadas ao código-fonte do projeto eram raramente testadas no tocante à sua integrabilidade com o restante do código. Assim, a possibilidade de existência de conflitos, *bugs* e incompatibilidades era ampliada, resultando em integrações complexas e arriscadas, além de gerar atrasos e retrabalho frequentemente.

Além disso, no modelo inicial, o processo de disponibilizar novas versões de um software para os usuários finais era burocrático, interferindo diretamente na produtividade do programador, que estava sujeito à uma série de etapas manuais. Essa abordagem, além

[5]Oh Good, The Leprechaun Films Are On Steam. Rock Paper Shotgun, 2016. Disponível em: `https://www.rockpapershotgun.com/steam-movie-streaming`. Acesso em: 25/09/2024.

de aumentar a incidência de erros humanos, gerava uma redução no *uptime*[6] da aplicação, uma vez que esse tipo de procedimento era realizado em janelas de manutenção, podendo ser programadas ou não.

Diante da necessidade de melhora dos aspectos relatados anteriormente, Thatikonda (2023) pontua que no início dos anos 2000 a ideia de validar o código com antecedência e realizar a entrega de mudanças mais frequentemente ficou mais comum. Com isso, surge o conceito de *Continuous Integration* (CI) que se trata de uma prática que consiste na integração automática e frequente de código em um mesmo repositório por parte dos desenvolvedores, permitindo que a detecção de possíveis problemas seja acelerada.

Um *pipeline* de CI normalmente é composto das etapas de *build*, empacotamento e testes automatizados, os quais devem gerar relatórios contendo resultados descritivos da operação, conforme apontado por Liu (2022). Além de verificar se o conjunto de alterações foram integradas corretamente ao projeto, essa última etapa pode ainda analisar aspectos como *linting* e percentual de cobertura do código por testes.

De acordo com Thatikonda (2023), o conceito de *Continuous Deployment* (CD) pode ser visto como uma extensão do processo de CI, em que procura-se aumentar o grau de automatização na jornada do desenvolvimento de software até o ambiente de produção. Essa abordagem permite que os usuários tenham acesso consistente à novas versões do software, além de diminuir a duração do ciclo de desenvolvimento, garantindo que atualizações sejam disponibilizadas com maior frequência.

Com isso, é possível determinar um pipeline de CI/CD por meio da junção das duas abordagens recém observadas, formando um poderoso conjunto de práticas que visam a automatização do processo de desenvolvimento de software ao máximo.

Figura 5: Pipeline de CI/CD. Fonte: mindbowser.com (2021)

[6]Uptime vs. Availability: How to measure and improve reliability. Pluralsight, 2024. Disponível em: `https://www.pluralsight.com/resources/blog/tech-operations/uptime-availability-metrics-app-reliability` da aplicação. Acesso em: 29/09/2024.

3 ANÁLISES E EXECUÇÃO

No contexto da fase de execução do projeto, fez-se necessário seguir algumas etapas, dentre elas, a análise estrutural da arquitetura, análise das tecnologias utilizadas e, por fim, a execução do processo de atualização tecnológica.

3.1 Análise estrutural da arquitetura

A etapa de análise estrutural da arquitetura do *launcher* foi de grande contribuição para assegurar que o processo de migração de paradigma de produto para serviço ocorresse sem maiores problemas. Essa investigação buscou entender o funcionamento da versão inicial do *launcher* mediante observação de seu *layout* e de suas funcionalidades principais, possibilitando a identificação de possíveis pontos de falha.

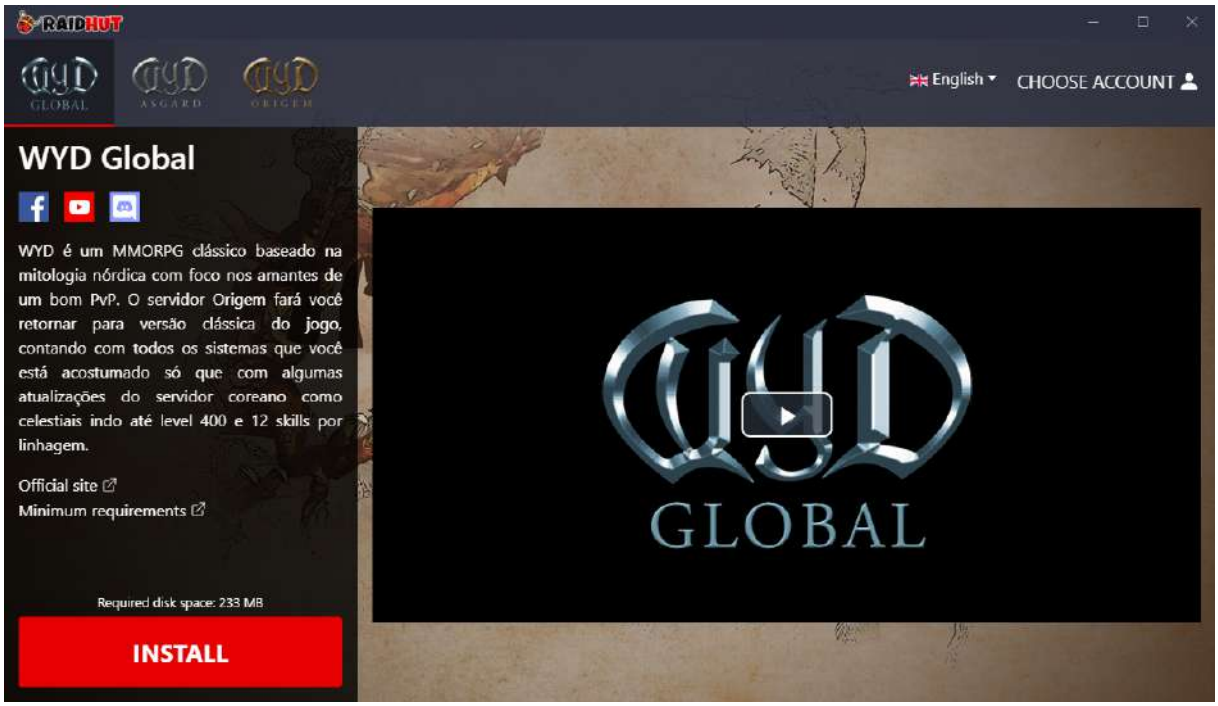

Figura 6: Visualização da versão inicial do *launcher*. Fonte: Raid Hut (2024)

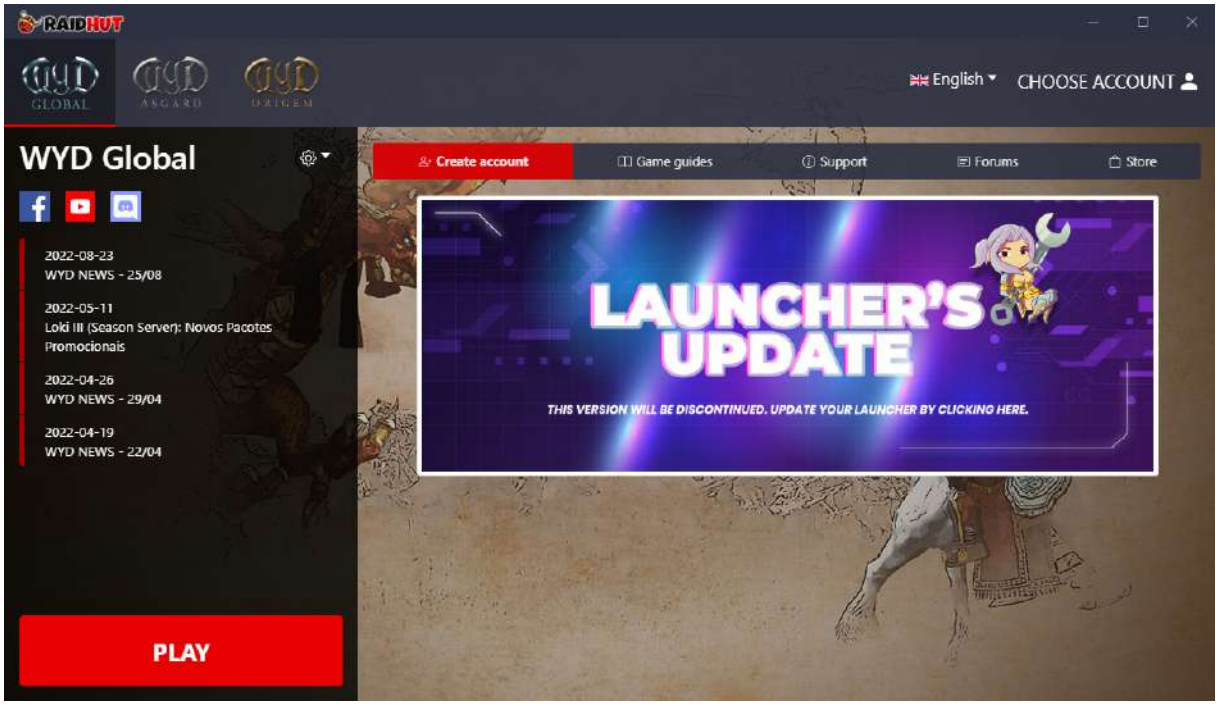

Figura 7: Visualização da versão inicial do *launcher* com um servidor instalado. Fonte: Raid Hut (2024)

Mediante observação das Figuras 6 e 7, pode-se destacar três seções principais: a barra superior, a seção lateral e a seção principal. Tais subdivisões serão analisadas a seguir.

3.1.1 Barra superior

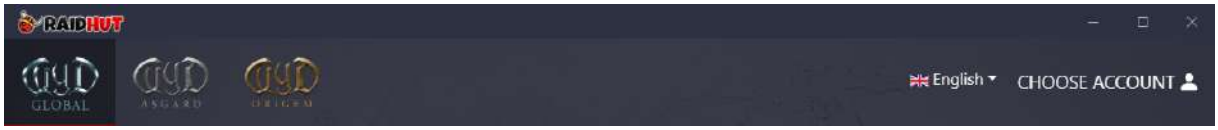

Figura 8: Visualização da barra superior do *launcher*. Fonte: Raid Hut (2024)

Conforme a imagem, trata-se de uma barra em que, no topo, é exibido o logo da Raid Hut, além de botões que permitem interagir com o aplicativo, sendo possível minimizar, alternar o tamanho da janela ou fechá-lo. Além disso, abaixo, existem três componentes a se observar: o seletor de servidor[7], o seletor de idioma e o botão para autenticação.

[7]Um servidor é uma instância resiliente e independente do mundo do jogo, compartilhada entre os jogadores.

```
"Global": {
  "name": "WYD Global",
  "size": "233 MB",
  "clientVersion": 0,
  "version": 583,
  "maintenance": {
    "next": "2024-09-26 11:00:00",
    "tz": "America/Fortaleza",
    "duration": 3
  },
  "banners": [
    {
      "link": "https://global.raidhut.com/eA5AwB",
      "image": "https://dashboard-rh.s3.amazonaws.com/banner/banner_24-09-26_17-02-20_Home_PT.png"
    },
    {
      "image": "https://download.raidhut.com.br/launcher/wyd/banners/btn_1.png",
      "link": "https://feedback.kersef.com"
    },
    {
      "image": "https://download.raidhut.com.br/launcher/wyd/banners/btn_2.png",
      "link": "https://feedback.kersef.com/en/knowledge-bases/21-faq-wyd-global"
    },
    {
      "image": "https://download.raidhut.com.br/launcher/wyd/banners/btn_3.png",
      "link": "https://feedback.kersef.com"
    }
  ],
```


Figura 9: Trecho do arquivo JSON que é lido pelo *launcher*. Fonte: Raid Hut (2024)

Nesse contexto, o *launcher* faz a leitura dos servidores disponíveis a partir de um arquivo *JavaScript Object Notation* (JSON) público, permitindo um maior reuso e modularidade do software, visto que novos servidores sejam adicionados com maior facilidade, sem a necessidade de criar uma nova versão da aplicação. A partir disso, informações cruciais de um servidor são alocadas dentro do JSON, como nome, versão, próxima manutenção agendada, links, entre outros.

O seletor de idioma, por sua vez, tem como função principal alterar o idioma do executável do jogo mediante a abertura via *launcher*, sendo possível escolher entre os idiomas inglês e português. Além disso, os textos dos botões do *launcher* eram traduzidos, bem como a forma em que as notícias exibidas na seção lateral era alterada, para que dessa forma, fossem resgatadas em sua versão traduzida.

Figura 10: Visualização do sistema de autenticação com falha. Fonte: Raid Hut (2024)

Por fim, o sistema de autenticação presente no *launcher* contava com 4 (quatro) maneiras diferentes de realizar o login, sendo elas: login via telefone, login via Apple, login via Facebook e login via Google. Todavia, notou-se que não estava sendo possível autenticar-se no *launcher*, devido ao fato de que o botão responsável pelo login não estava renderizando a aba com as opções de login corretamente, ficando em um estado de carregamento eterno.

3.1.2 Seção lateral

Figura 11: Visualização da seção lateral do *launcher*. Fonte: Raid Hut (2024)

De acordo com a Figura 11, é possível visualizar logo ao topo o nome do servidor selecionado pelo usuário durante sua escolha através do seletor de servidor. Além disso, são exibidos ícones de mídias sociais referentes ao servidor específico e também as notícias direcionadas ao mesmo, as quais são também exibidas no *website* oficial do jogo.

No entanto, durante o processo de análise desse componente, percebeu-se que na versão em inglês, as notícias não estavam sendo resgatadas no idioma correto. Além disso, as notícias exibidas eram estáticas, não eram modificadas ou sincronizadas com as notícias presentes no site oficial da Raid Hut. Com isso, notou-se que o problema trata-se de um ponto de falha a ser corrigido, pois gerava-se uma notável inconsistência na experiência de usuário.[8]

Finalmente, na parte inferior da Figura 11 é visto um botão vermelho, que pode conter dois textos diferentes: "Instalar"ou "Jogar". A opção de instalação surge para o usuário quando os arquivos do servidor selecionado na barra superior não encontram-se instalados em sua máquina. Por outro lado, a opção de jogar é exibida para o jogador caso o servidor esteja devidamente instalado e atualizado no dispositivo. Durante a etapa de análise das tecnologias pré-existentes desse trabalho, o processo de download de patches que ocorre durante a instalação será abordado brevemente.

[8]Why Consistency Across Platforms Matters for User Experience. LoopQA, 2023. Disponível em: `https://www.workwithloop.com/blog/why-consistency-across-platforms-matters-for-user-experience`. Acesso em: 25/09/2024.

3.1.3 Seção principal

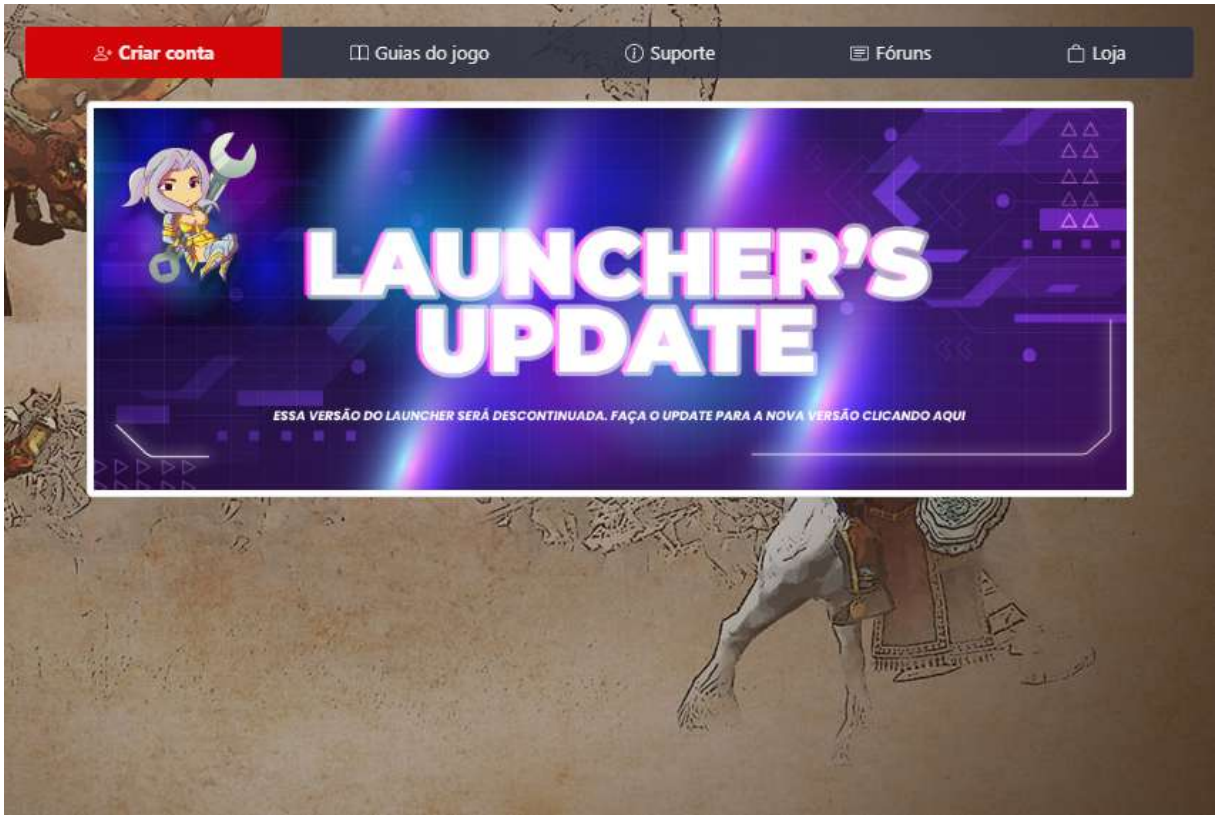

Figura 12: Visualização da seção principal do *launcher*. Fonte: Raid Hut (2024)

Diante do exposto na Figura 12, é possível dividir a seção principal do *launcher* em duas porções importantes: o navegador de utilitários e o *banner* de evento. O navegador de utilitários permite com que o usuário acesse com maior rapidez uma das páginas representadas pelo texto do botão. De acordo com SHNEIDERMAN e PLAISANT (2010), interfaces efetivas geram sentimento positivos de sucesso na comunidade de usuários, não tornando o usuário onerado pelo layout, lhes permitindo prever o que acontecerá a seguir.

Entretanto, os botões presentes na seção principal que deveriam redirecionar para páginas específicas a depender do idioma, encontravam-se com problemas. Em particular, somente páginas em inglês eram entregues ao usuário.

Por fim, a região do *banner*, é representada por uma imagem que corresponde ao evento que está em execução no servidor em questão. Ao clicá-la, o usuário será redirecionado para a notícia correspondente, onde será possível visualizar todos os detalhes referentes ao evento em questão, melhorando a imersão do usuário no mundo do jogo.

Pontos de falha	Descrição
1	Falha no processo de autenticação
2	Inconsistência nas notícias internacionais
3	Inconsistência nos botões de redirecionamento

Quadro 1: Pontos de falha percebidos

3.2 Análise tecnológica

Uma questão essencial no processo de melhoria do *launcher* foi a análise das tecnologias utilizadas na aplicação, isto é, ferramentas e bibliotecas pré-existentes no projeto. Al-Rubaye (2020) destaca que a crescente necessidade dos desenvolvedores em oferecer serviços mais consistentes, confiáveis e seguros gera uma tendência em substituir as bibliotecas antigas por novas. Além disso, a ausência de atualizações nessas tecnologias cria uma limitação no processo de inovação da aplicação, pois frequentemente gera um conflito de versão entre pacotes e bibliotecas.

Ademais, é necessário identificar quais ferramentas e tecnologias serão necessárias para trazer novas funcionalidades ao projeto. A adição de elementos ao *launcher*, busca, além de tornar a experiência do jogador mais imersiva, melhorar a coleta de dados por parte da empresa. Com isso, torna-se possível o investimento em campanhas mais direcionadas à públicos específicos, uma vez que a dor do usuário está melhor interpretada.

3.2.1 Tecnologias pré-existentes

Inicialmente, foram identificadas as principais ferramentas e bibliotecas empregadas na versão inicial do *launcher*, destacando o Electron como a ferramenta principal responsável por possibilitar a execução da aplicação como um software de *desktop*. Durante a análise, constatou-se que a versão utilizada do Electron era a 7.1, enquanto a versão 22.3 já estava disponível para uso. Além disso, observou-se que a biblioteca de *frontend React*, empregada na interface, estava na versão 16.12, em comparação à versão mais recente, 18.2.

Somado a isto, no contexto de distribuição e empacotamento do *launcher*, a fim de gerar um instalador, utilizava-se a ferramenta Electron Builder. Embora ainda fosse amplamente utilizada para esse intuito, notou-se a existência de uma outra ferramenta que prometia entregar resultados parecidos com menor configuração inicial, o Electron Forge, que logo foi escolhido como substituto para o seu similar.

Além disso, passou-se a utilizar o framework Squirrel.Windows integrado com o Electron Forge, permitindo a criação e distribuição de instaladores velozes, par os quais é exigido o mínimo esforço por parte do usuário.

Nota-se ainda que, em sua versão inicial, o *launcher* fazia uso de uma biblioteca chamada electron-download-manager para realizar o processo de download dos patches de jogo. Após pesquisas, percebeu-se que a biblioteca estava obsoleta, com cerca de 5 anos sem manutenção e pouquíssima aceitabilidade entre os desenvolvedores ao redor do mundo.

Com isso, uma abordagem mais tradicional foi implementada, substituindo o electron-download-manager pela biblioteca axios[9]. Com atualizações constantes e uma base expressiva de usuários, a utilização da nova biblioteca gerou uma notável melhora no processo anteriormente descrito, evitando travamentos e perda de pacotes que ocorriam em alguns casos.

3.2.2 Atualizações de bibliotecas

Algumas tecnologias foram atualizadas, dentre elas o serviço Firebase Authentication, presente na plataforma Firebase. Utilizando esse serviço, foi possível reconstruir o sistema de autenticação do *launcher*, permitindo que os usuários façam devidamente o login.

Usuário autenticados, passaram então a ter acesso a um sistema de recompensas dentro de jogo, mediante a participação em alguma pesquisa realizada pela equipe Raid Hut, acessíveis somente através do *launcher*. Do ponto de vista empresarial, a coleta de respostas advindas da base de jogadores é de extrema importância, observada a possibilidade de adoção de medidas estratégicas baseada na opinião do usuário.

Além do mecanismo de pesquisas, foi possível obter feedbacks indiretos do jogador, por meio de um sistema de análise comportamental. Para tanto, implementou-se a biblioteca react-microsoft-clarity, permitindo a utilização do Microsoft Clarity, no qual, de acordo com Kajima e Kikuchi (2023) é possível observar a interação do usuário com a aplicação, visualizando os principais pontos de atenção, sendo de grande importância no processo de melhoria de experiência do usuário e de evolução de software.

[9]Getting Started. Axios, 2024. Disponível em: `https://axios-http.com/docs/intro`

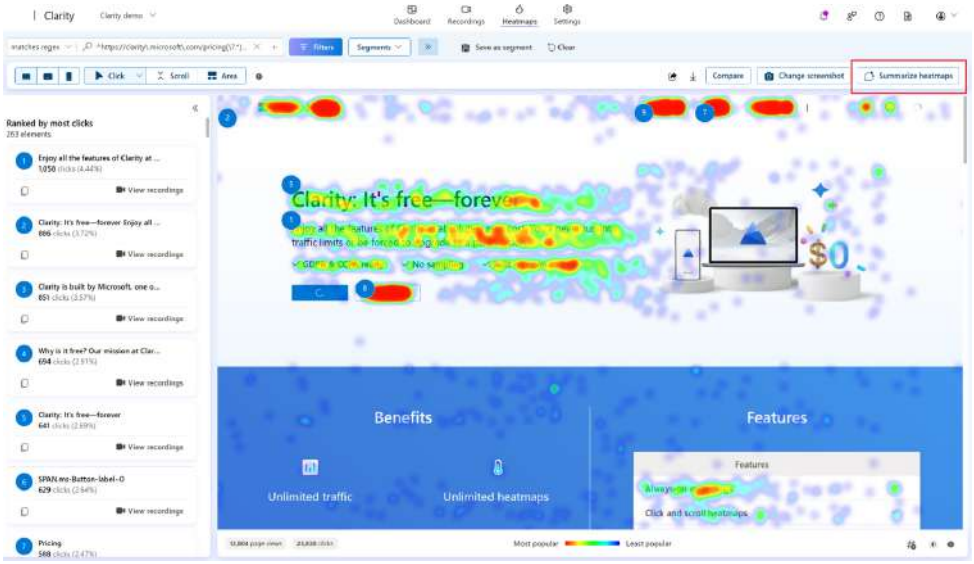

Figura 13: Exemplo de visualização de mapa de calor em aplicação com Clarity. Fonte: Microsoft (2024)

Outra adição importante ao projeto se tratou do Jest como ferramenta para a implementação de testes unitários. Com sua utilização, foi possível tornar o código mais estável durante o desenvolvimento, melhorando a detecção de possíveis problemas antes que chegue ao usuário final. Além disso, o uso de testes unitários evita que problemas que aconteceram anteriormente voltem a acontecer no ambiente de produção, aumentando a qualidade de vida para o software enquanto serviço.

3.3 Execução do processo de atualização tecnológica

A etapa de execução da atualização tecnológica consiste em adotar medidas práticas relacionadas ao projeto, de modo que sejam implementadas as configurações, melhorias e testes que garantam a sua modernização.

3.3.1 Configurações iniciais

Inicialmente, para a execução do processo de atualização tecnológica, foi criado um novo projeto Electron, seguindo o passo a passo disponibilizado na própria documentação do framework.[10] Dessa forma, foi possível criar um projeto simples, mantido em um repositório privado no GitHub contendo somente os arquivos necessários e com as bibliotecas atualizadas.

Posteriormente, realizou-se a configuração inicial do Electron Forge, através da

[10]Quick Start. ElectronJS, 2023. Disponível em: `https://www.electronjs.org/docs/latest/tutorial/quick-start`. Acesso em: 16/09/2024.

criação do arquivo forge.config.ts, no qual são definidas as preferências de empacotamento[11] e distribuição do programa.

```
makers: [
  new MakerSquirrel({}),
  {
    name: "@electron-forge/maker-squirrel",
    config: {
      setupExe: "WYD_Setup.exe",
      certificateFile: "./raid.pfx",
      loadingGif: "./src/assets/logo.gif",
      iconUrl: "https://dashboard-rh.s3.amazonaws.com/launcher/logowydbr.ico",
      setupIcon: "./src/assets/logowydbr.ico",
      remoteReleases:
        "https://dashboard-rh.s3.us-east-1.amazonaws.com/launcher/",
    },
  },
],
```

Figura 14: Trecho de código referente à configuração do Electron Forge. Fonte: Raid Hut (2024)

Dessa forma, passou-se dentro das opções de configuração, o Squirrel.Windows como sendo o mecanismo de empacotamento para a aplicação. Além disso, definiu-se através de parâmetro um *bucket* do *Amazon Simple Storage Service* (S3) para ser o repositório remoto que guardará todas as versões remotas do *launcher*.

Com isso, iniciou-se o processo de configuração de um pipeline CI/CD, fazendo uso da plataforma de automação GitHub Actions com o intuito de automatizar algumas etapas essenciais, tais como o build do projeto e a execução de testes. Devido à utilização do GitHub como repositório de controle de versão, o uso de GitHub Actions é vantajoso devido à sua integração nativa e proximidade com o repositório, tornando todo o processo mais simples, conforme observado por Graciano e Kakinohana (2023).

```
- run: yarn install --network-timeout 1000000
- run: yarn make
- name: Configure AWS Credentials
  uses: aws-actions/configure-aws-credentials@v1
  with:
    aws-access-key-id: ${{ secrets.AWS_KEY_ID }}
    aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
    aws-region: us-east-1

- name: Copy files to the s3 website content bucket
  run: aws s3 sync out\make\squirrel.windows\x64 s3://dashboard-rh/launcher
```

Figura 15: Trecho de código referente ao pipeline de CI/CD. Fonte: Raid Hut (2024)

Como pode ser observado na Figura 15, inicialmente executa-se o comando yarn

[11] Entendendo o empacotamento e distribuição de app com Electron — Parte I. Medium, 2019. Disponível em: `https://medium.com/@JuliaBrazolim/entendendo-o-empacotamento-e-distribui%C3%A7%C3%A3o-de-app-com-electron-parte-i-4508b4ba09a6`. Acesso em: 16/09/2024.

install, para o qual, por meio do gerenciador de pacotes Yarn, busca-se instalar todas as dependências do projeto previstas no arquivo package.json. Posteriormente, é utilizado o comando yarn make que se trata de um *script* correspondente ao electron-forge make, cujo objetivo é empacotar a aplicação, além de gerar a distribuição por meio do squirrel-windows.

Posteriormente, são definidas as credenciais em que o GitHub Actions deve usar ao tentar realizar ações no S3. Tais credenciais foram guardadas através da ferramenta Secrets que o GitHub disponibiliza, permitindo que chaves de *Application Programming Interface* (API), senhas e *tokens* de acesso fiquem salvos de forma segura dentro de um repositório.

A partir das configurações feitas no projeto, ao final do processo de deploy, os arquivos gerados através do *build* realizado pelo Electron Forge são copiados para um bucket do S3. Esse bucket tem por finalidade servir de repositório para que a busca por atualizações do Electron possa posteriormente consultá-lo, conforme é possível observar na imagem abaixo:

Figura 16: Trecho de código referente à checagem de atualizações do *launcher*. Fonte: Raid Hut (2024)

Com base na Figura anterior, pode-se notar que inicialmente é feita a verificação se o programa está rodando em sua versão final (através do .exe gerado via empacotamento). Em caso afirmativo, é definido um endereço de onde essa checagem deverá se basear. A *flag* faz menção a uma variável que busca entender se o programa foi recém-atualizado, para evitar a busca por novas atualizações de forma desnecessária.

3.3.2 Realocação dos componentes

Durante o processo de migração da versão legada do *launcher*, vários arquivos puderam ser reaproveitados, alguns dos quais eram arquivos escritos em *TypeScript*, que possuem a extensão `.ts` e `.tsx`. Dessa forma, foi realizada uma abordagem criteriosa, analisando-se os arquivos individualmente para ver a necessidade de seu uso, podendo

assim deixar de ser usado na nova aplicação. Esse processo visa evitar *dead code*, economizando mais memória para a execução além de um código mais limpo e organizado.

Além disso, durante a realocação dos componentes, foi possível revisar todas as dependências externas, evitando-se a inclusão de bibliotecas que não seriam utilizadas no contexto da nova aplicação. Com isso, o projeto apresentou melhora de desempenho, além de facilitar o processo de manutenção devido à diminuição de código extra.

3.3.3 Implementação de novas funcionalidades e correção de problemas

O modelo de autenticação da aplicação foi totalmente remodelado, visando um maior controle sobre a identidade real dos usuários e aumentando a segurança por parte da empresa. Dessa forma, foi implementado um sistema de autenticação utilizando *Quick-Response Code* (QR Code), em que o usuário deverá realizar o devido escaneamento a fim de concluir o processo de login.

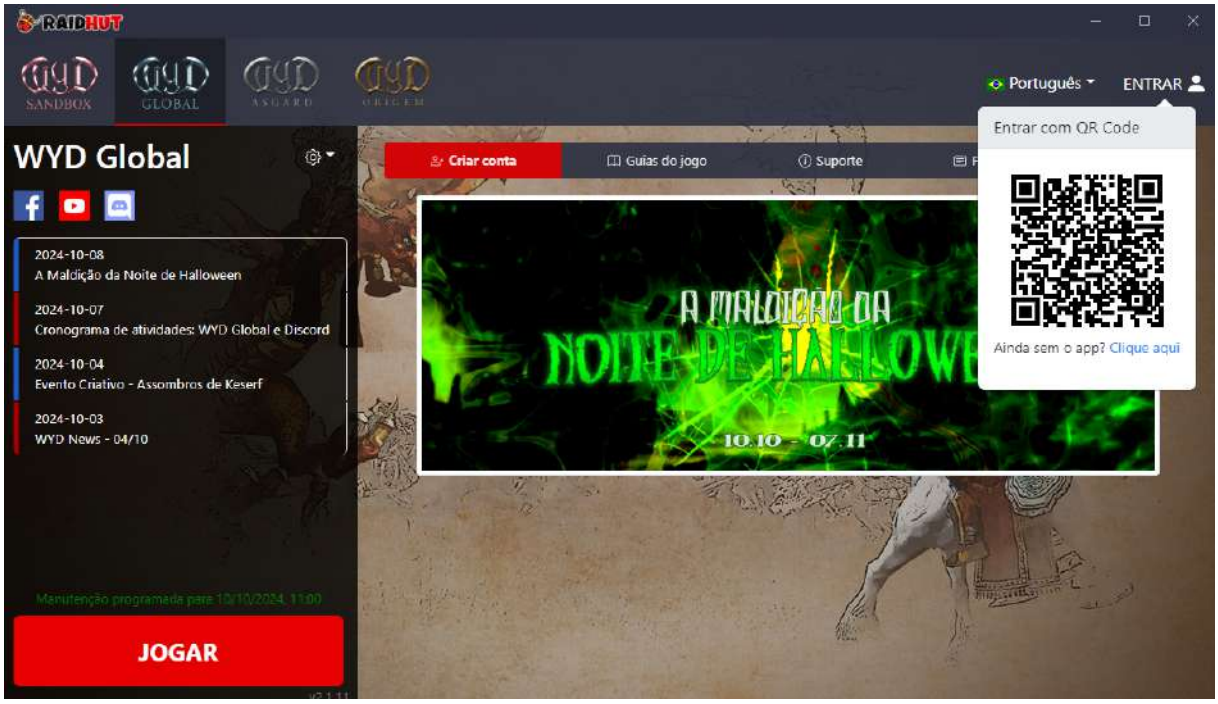

Figura 17: Visualização do *launcher* contendo um QR Code para login. Fonte: Raid Hut (2024)

Inicialmente, ao abrir o *launcher* sem estar autenticado, o usuário tem a opção de realizar o login na aplicação, clicando no devido botão exibido anteriormente. Assim, um *token* gerado por uma API interna será exibido ao usuário em formato de QR Code, enquanto o *launcher* checará constantemente via API até que esse token esteja associado à um número de telefone.

Um detalhe importante trata-se do fato de que o *token* é gerado de forma segura,

por meio de um processo interno que utiliza chaves secretas e interpolação de *strings* para a sua geração. Logo, no momento da leitura do QR Code, verifica-se a legitimidade do *token*, evitando que ações maliciosas por parte do usuário possam ser realizadas.

Sabendo disso, o usuário deverá utilizar o *Raid Hut App*, um aplicativo próprio para dispositivos móveis em que, mediante devida autenticação via número de telefone por meio do Firebase, existirá a opção de leitura de QR Code. Durante esse processo de leitura, o aplicativo verifica se o *token* em questão é válido e o associa ao número de telefone no jogador em um banco de dados relacional.

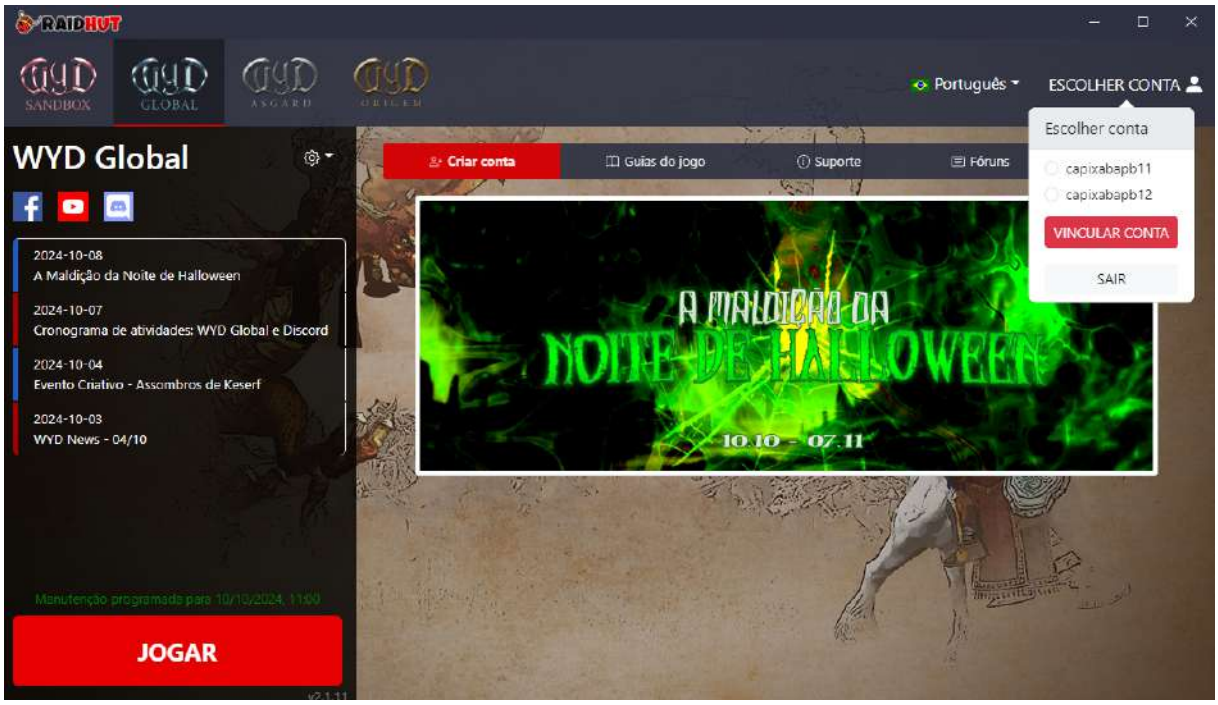

Figura 18: Visualização do *launcher* com usuário autenticado. Fonte: Raid Hut (2024)

Após o *launcher* constatar que o *token* gerado está associado à um usuário do Firebase (por meio do número de telefone), o usuário ficará devidamente autenticado na aplicação. Dessa forma, é possível identificar unicamente um utilizador, tanto na plataforma do *launcher* como pelo aplicativo, permitindo que informações cruciais sejam extraídas, tais como contas vinculadas ao número de telefone e último registro de participação em eventos.

Outra adição importante, trata-se da implementação do Microsoft Clarity ao projeto que consistiu na instalação da biblioteca react-microsoft-clarity, além de incorporar o *script* de inicialização e consentimento da ferramenta ao código da aplicação. Nota-se que o consentimento do usuário foi previamente solicitado durante a criação de conta, na qual é solicitada a leitura da política de privacidade da empresa, em que, é explicitada a coleta de dados de usuário na plataforma do game.

```
clarity.init("ID do projeto do Clarity");
clarity.consent();
```

Figura 19: Exemplo de utilização dos métodos principais da biblioteca. Fonte: Raid Hut (2024)

Dessa forma, é inicialmente chamado um método responsável pela inicialização do Clarity no projeto, passando-se como parâmetro o ID do projeto registrado na plataforma do Microsoft Clarity e posteriormente usa-se um método que determina que há consentimento do usuário para utilização da ferramenta.

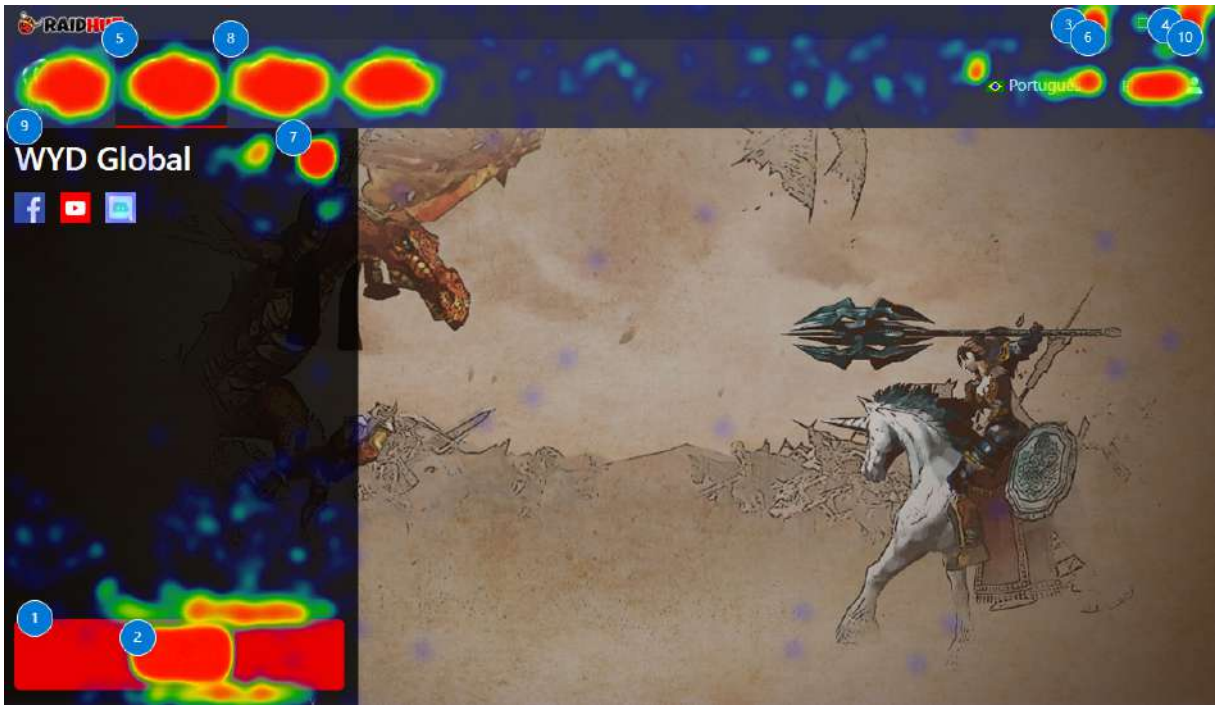

Figura 20: Visualização do mapa de calor do *launcher* no Microsoft Clarity. Fonte: Raid Hut (2024)

Conforme visto na imagem anterior, percebe-se que o Microsoft Clarity fornece uma visualização detalhada do comportamento dos usuários no *launcher*, permitindo a análise de áreas mais acessadas e com maior interação. Essa informação é importante para entender quais elementos são mais atrativos para o usuário, de modo que seja possível trabalhar em ajustes que visem melhorar a experiência de usuário e aumentar o engajamento.

Por conseguinte, deu-se início a uma investigação acerca do ponto de falha referente às notícias internacionais não estarem sendo atualizadas na seção lateral do *launcher*. De forma similar à alguns outros componentes da aplicação, as informações sobre as notícias à serem disponibilizadas são obtidas por meio do arquivo JSON utilizado pelo *launcher*,

onde nesse caso, aponta-se para um *endpoint* próprio da companhia que deve cumprir essa função.

Após análise, constatou-se que o código presente na API referente ao *endpoint* citado havia sido modificado, resultando em uma dessincronia com a versão inicial do *launcher*. O problema consistia no fato de que originalmente, para acessar as notícias da comunidade internacional, o *launcher* enviava o parâmetro "tag"por meio de uma requisição do tipo *GET* para o *endpoint* supracitado. No entanto, após a alteração, o envio desse parâmetro se tornou desnecessário. Assim, o código do *launcher* foi ajustado para refletir essa mudança, parando de enviar o parâmetro em questão e garantindo a correta sincronização com a nova versão do *endpoint*.

Por fim, a mesma solução foi aplicada aos botões de redirecionamento presentes na seção principal do launcher, permitindo que usuários fossem redirecionados corretamente para as páginas de seu idioma de escolha.

3.3.4 Implantação de testes unitários

Enfim, fez-se necessário garantir que futuras alterações não introduzissem novos erros e que a funcionalidade de todos os componentes se mantivesse intacta. Para isso, foram implantados testes unitários utilizando a biblioteca Jest e a React Testing Library, que permitem a criação de uma base de testes sólida para aplicações em React.

```
describe("Combined Component Tests", () => {
  describe("MTitleBar Component", () => {
    it("renders correctly", () => {
      const { getByTestId } = render(<MTitleBar />);
      expect(getByTestId("close-button")).toBeInTheDocument();
      expect(getByTestId("minimize-button")).toBeInTheDocument();
      expect(getByTestId("maximize-button")).toBeInTheDocument();
      expect(getByTestId("double-click-area")).toBeInTheDocument();
    });

    it("sends the correct IPC messages on user interactions", () => {
      const { ipcRenderer } = require("electron");
      const { getByTestId } = render(<MTitleBar />);

      fireEvent.click(getByTestId("close-button"));
      expect(ipcRenderer.send).toHaveBeenCalledWith("close");

      fireEvent.click(getByTestId("minimize-button"));
      expect(ipcRenderer.send).toHaveBeenCalledWith("minimize");

      fireEvent.click(getByTestId("maximize-button"));
      expect(ipcRenderer.send).toHaveBeenCalledWith("maximize");

      fireEvent.doubleClick(getByTestId("double-click-area"));
      expect(ipcRenderer.send).toHaveBeenCalledWith("maximize");
    });
  });

});
```

Figura 21: Trecho de código referente ao teste da barra superior. Fonte: Raid Hut (2024)

Conforme visto na imagem, é apresentado um trecho de código demonstrando a utilização das bibliotecas anteriormente citadas. Observa-se a presença de métodos advindos do Jest, *describe* e *it*, que buscam respectivamente especificar o escopo de um grupo de testes e verificar se um componente renderiza conforme o esperado.

Nesse contexto, durante o teste, é utilizado o método *render*, advindo da biblioteca React Testing Library, que permite a renderização de um componente em um ambiente simulado, permitindo interações e asserções. Além disso, o Jest fornece o método *expect*, que traz a possibilidade de verificar se determinadas condições são válidas.

Finalmente, é feito um teste primário de renderização, verificando-se a existência de determinados componentes na aplicação, consolidando-se através do método *expect*. Posteriormente, um teste é realizado para garantir que determinadas ações de clique por parte do usuário sejam corretamente tratadas entre o processo de renderização e o processo principal do Electron.

3.4 Resultados

3.4.1 Resultados gerais

Diante do exposto, na percepção da equipe responsável pelo launcher, o processo de atualização tecnológica realizado no *launcher* trouxe significativos benefícios para diferentes *stakeholders*, conforme é possível observar no quadro a seguir:

Stakeholder	**Benefícios Observados**
Organização (Raid Hut)	- Redução no custo de manutenção e melhoria
Cliente (Time de Produto)	- Maior liberdade em solicitação de funcionalidades - Maior rapidez na entrega de suas solicitações
Usuário (Jogador)	- Apenas necessita de uma instalação de software - Acesso à novidades com maior rapidez - Melhor desempenho e segurança
Provedor (Time de Desenvolvimento)	- Redução no tempo investido em processos manuais - Redução de intervenção humana no deploy do sistema - Aumento na qualidade do código

Quadro 2: Benefícios observados para cada *stakeholder*

Além disso, a utilização de CI/CD no projeto permitiu que a incidência de problemas de incompatibilidade de ambiente entre os desenvolvedores[12] diminuísse em grande escala, devido aos procedimentos executados durante o pipeline de CI que ocorrem em um ambiente isolado. Ao final do processo de atualização tecnológica, foi constatada a correção de diversos *bugs*, anteriormente listados, além da inclusão de novas funcionalidades, como o novo método de autenticação.

Enquanto na versão inicial do *launcher* o processo de *deploy* era totalmente manual, ao automatizá-lo houve uma evidente redução no tempo necessário para o *deploy*, permitindo que as atualizações cheguem mais rapidamente aos jogadores. Essa abordagem reduz o custo investido no processo, ao economizar o tempo de pessoal anteriormente designado para tal, possibilitando uma mudança de alocação conforme as necessidades da empresa.

A evolução observada na aplicação em questão tem como referência a mudança de paradigma estabelecida durante o surgimento da plataforma Steam, que incorporou novas práticas ao método de distribuição de jogos. O software da Valve Coporation sofreu mudanças ao longo do tempo, melhorando sua performance e entregando constantemente um software com maior qualidade ao usuário final.

[12]But... it works on my machine.... Medium, 2023. Disponível em: `https://medium.com/@josetecangas/but-it-works-on-my-machine-cc8cca80660c`

3.4.2 Análise de percepção

É possível entender melhor a eficácia de uma solução por meio de uma análise de percepção dos envolvidos. Nesse contexto, realizou-se uma pesquisa com dois grupos distintos da empresa: o time de produto e o time de desenvolvimento. Para cada grupo, foram realizadas 5 (cinco) questões de múltipla escolha, acerca dos novos processos e automações implementadas no *launcher*.

Além disso, para a realização desse questionário foi utilizada a escala Likert[13], em sua versão de 5 pontos. De acordo com Joshi (2015), a escala Likert é uma ferramenta essencial para quantificar atitudes, percepções e opiniões humanas, elementos que são subjetivos e difíceis de medir utilizando métodos tradicionais.

Durante a pesquisa com o time de desenvolvimento, avaliou-se respectivamente as seguintes métricas: diminuição do tempo gasto em atividades repetitivas e manuais, maior agilidade na integração de novas funcionalidades, maior segurança no processo de *deploy*, maior confiança na qualidade das versões mediante implementação dos testes e maior nível de colaboração entre as equipes, após as automações. Para tanto, houveram 4 (quatro) participantes, que deveriam dar suas opiniões a respeito de questões variando em uma escala de "discordo totalmente"até "concordo plenamente".

[13]O que é escala Likert e como aplicá-la na pesquisa? MindMiners, 2023. Disponível em: `https://mindminers.com/blog/entenda-o-que-e-escala-likert`

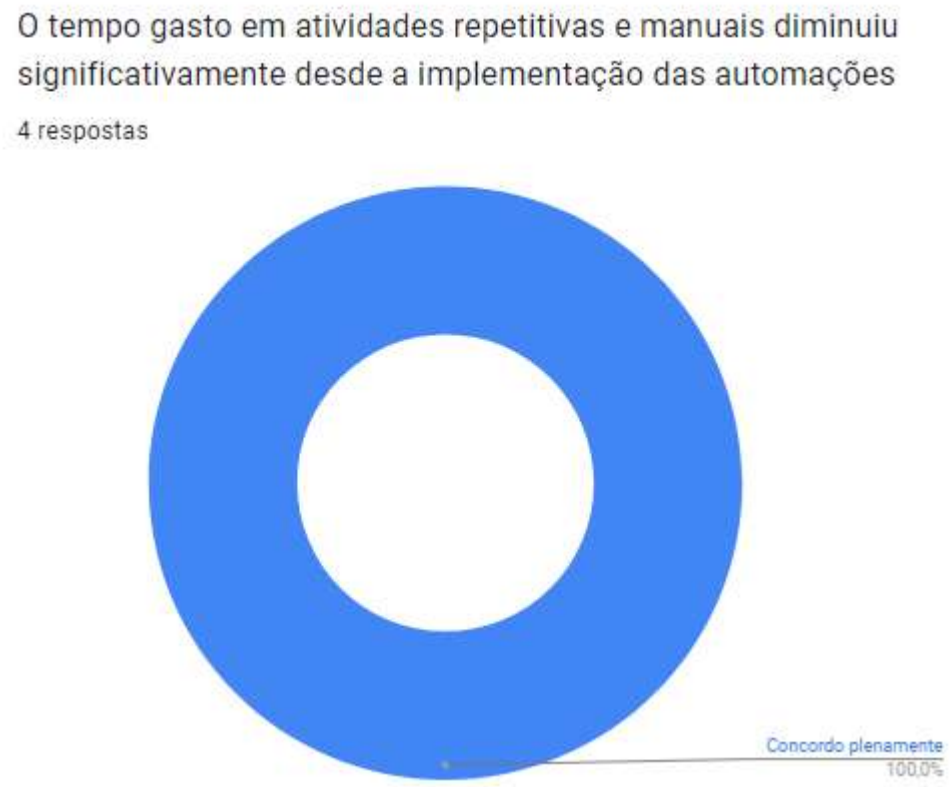

Figura 22: Avaliação do tempo gasto em atividades manuais. Fonte: Autor (2024)

Figura 23: Avaliação da agilidade na integração de novas funcionalidades. Fonte: Autor (2024)

Figura 24: Avaliação do aumento na segurança do processo de deploy. Fonte: Autor (2024)

Figura 25: Avaliação do aumento na confiança da qualidade das versões. Fonte: Autor (2024)

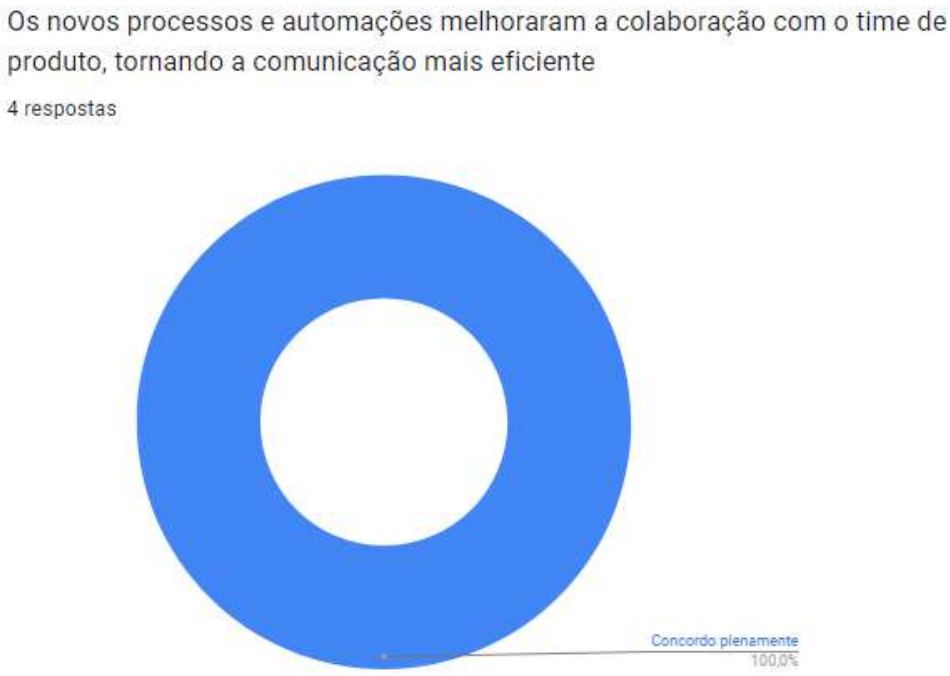

Figura 26: Avaliação do aumento na colaboração entre as equipes. Fonte: Autor (2024)

Mediante observação dos gráficos apresentados, é possível ver que houve quase uma unanimidade dentro do time de desenvolvedores, no que se refere à concordar plenamente com as proposições dadas. Exceto no caso da Figura 25, na qual 25% (1 desenvolvedor) concorda apenas parcialmente com o que foi proposto.

Por fim, na pesquisa realizada com o time de produto, as métricas avaliadas foram, respectivamente: melhora da entrega de valor, maior agilidade na entrega de recursos, aumento na colaboração entre as equipes, redução na quantidade de erros que chegam até o usuário e número cada vez menor de reclamações sobre o *launcher*.

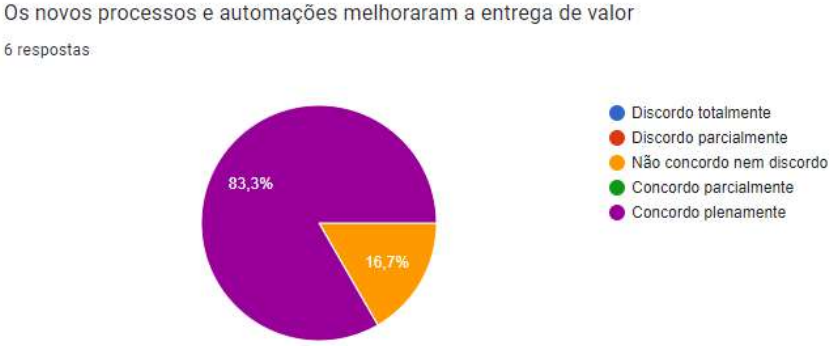

Figura 27: Avaliação da melhora de entrega de valor. Fonte: Autor (2024)

Conforme apresentado, cinco participantes (83,3%) concordam plenamente que os

novos processos e automações melhoraram a entrega de valor, enquanto um participante (16.7%) concorda apenas de forma parcial.

Figura 28: Avaliação da agilidade na entrega de novos recursos. Fonte: Autor (2024)

Conforme visto, quatro participantes (66,7%) concordam plenamente que as automações tornaram a entrega de novos recursos mais ágil, um participante (16,7%) concorda apenas de forma parcial e finalmente, um participante (16,7%) não concorda nem discorda.

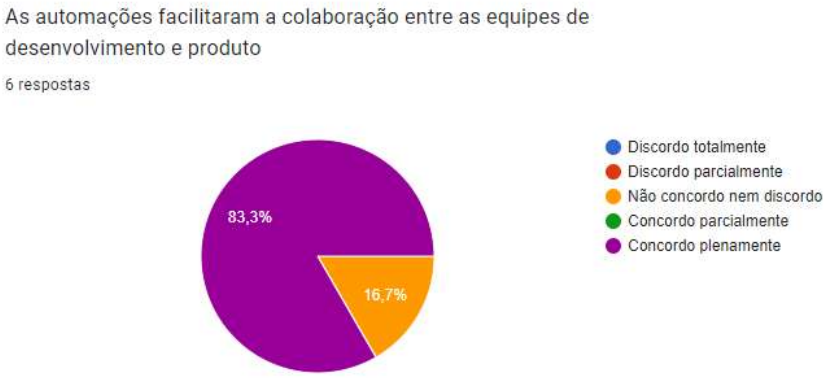

Figura 29: Avaliação do aumento na colaboração entre as equipes. Fonte: Autor (2024)

Conforme visto, cinco participantes (83,3%) concordam plenamente que as automações facilitaram a colaboração entre as equipes de desenvolvimento e produto, enquanto um participante (16,7%) concorda apenas de forma parcial.

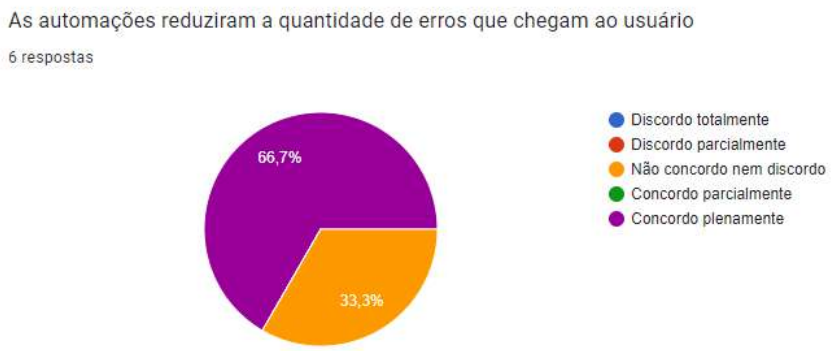

Figura 30: Avaliação da redução na quantidade de erros que chegam ao usuário. Fonte: Autor (2024)

Conforme visto, quatro participantes (66,7%) concordam plenamente que as automações reduziram a quantidade de erros que chegam ao usuário, enquanto dois participantes (33.3%) concordam apenas de forma parcial.

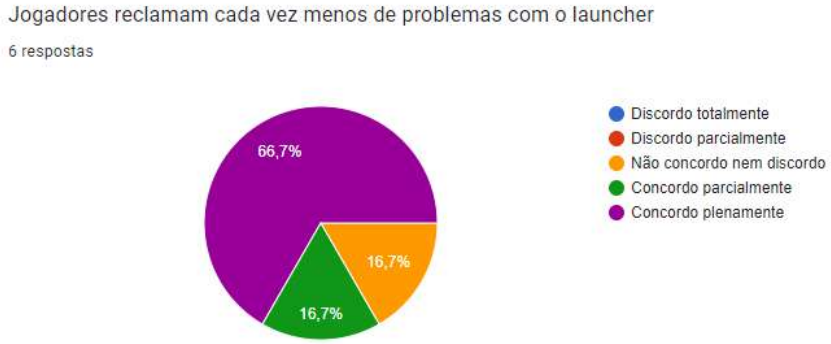

Figura 31: Avaliação da constante diminuição do número de reclamações referente ao *launcher*. Fonte: Autor (2024)

Conforme visto, quatro participantes (66,7%) concordam plenamente que os jogadores reclamam cada vez menos de problemas com o *launcher*, um participante (16,7%) concorda apenas de forma parcial e finalmente, um participante (16,7%) não concorda nem discorda.

4 CONCLUSÕES E TRABALHOS FUTUROS

A implementação de novas tecnologias e correção de problemas é crucial para a longevidade de um software. Segundo Lehman (1980), os sistemas evoluem continuamente, tornando-se mais complexos com o tempo, e a manutenção envolve correção de erros, otimização e adoção de novas tecnologias para garantir a eficiência e relevância do software. Assim, relevantes tecnologias foram inseridas ao projeto, tais como o Microsoft Clarity e o próprio CI/CD, além de correções de bugs encontrados durante o processo de análise estrutural.

Entre as etapas fundamentais do trabalho apresentado neste relatório técnico, é possível destacar a análise estrutural da arquitetura do *launcher* e a identificação de pontos de falha na versão inicial, que serviram como base para as melhorias propostas. No entanto, o entendimento e adaptação à novas tecnologias se trataram de dificuldades percebidas ao decorrer desse estudo, por exigirem um maior investimento de tempo e esforço para a correta aplicação.

Nesse sentido, a migração de paradigma de produto para serviço por meio da implementação de práticas de CI/CD foi um passo importante na atualização tecnológica da aplicação, pois se trata de uma abordagem que traz benefícios para todos os envolvidos. As vantagens vão desde o desenvolvedor que terá maior proximidade à aplicação, lhe permitindo integrar novas funcionalidades de forma mais ágil, até o usuário final o qual terá menos obstáculos durante o uso do software, gerando uma experiência mais satisfatória.

Com relação à trabalhos futuros, é importante que as estratégias e conceitos aplicados sejam continuamente evoluídos para atender aos padrões de qualidade esperadas da Raid Hut. Em específico, o *launcher* visto deve estar em constante processo de melhoria, principalmente pela adição de ferramentas e componentes que expandem o conhecimento sobre o tipo de comportamento dos usuários e seu nível de satisfação.

Além disso, uma abordagem plausível seria a utilização de AI na análise dos dados coletados pelo Microsoft Clarity, no qual, com base em determinados padrões de uso dos usuários, determinados problemas e aperfeiçoamentos poderiam ser identificados e sugeridos para a empresa. Nessa perspectiva, o aprimoramento contínuo da aplicação seria reforçado, atendendo mais precisamente a necessidade dos utilizadores que possivelmente despercebidas por análises humanas ou *feedback* dos usuários.

REFERÊNCIAS

[1] ARORA, Krishan. The Gaming Industry: A Behemoth With Unprecedented Global Reach. Forbes Agency Council, 2023. Disponível em: <https://www.forbes.com/councils/forbesagencycouncil/2023/11/17/the-gaming-industry-a-behemoth-with-unprecedented-global-reach>. Acesso em: 20 out. 2024.

[2] CARVALHO, B.; SOARES, M.; NEVES, A.; SOARES, G.; LINS, A. Virtual Reality devices applied to digital games: a literature review. **Ergonomics in Design**, pp. 125-141, September 2016.

[3] FICHEMAN, I. K; AUGUSTO, R. L.; SILVA, D. L. S.; LOPES, R. D. D.; ZUFFO, M. K. **GP Brasil VR: uma Corrida de Carros em Realidade Virtual**. In: V Brazilian Symposium on Computer Games and Digital Entertainment. Anais V Brazilian Symposium on Computer Games and Digital Entertainment. São Paulo: Escola Politécnica da Universidade de São Paulo, 2006.

[4] GEORGE, J. B. P. D. O. D. F. **The impact of Software as a Service in Software Piracy**: Has the change in the distribution and sale of software provided not only an accurate answer against software piracy but also an increase in consumer-value?. 2013. Tese de Mestrado: Universidade Católica Portuguesa – Lisboa, Portugal. 2013.

[5] KANERVA, S. **Lateral Movement Restriction in Advanced DevOps Environments**. 2024. Tese de Mestrado: Aalto University – Espoo, Finlândia. 2024.

[6] ROWE, J. **Valve's Contribution to the Gaming World**. In: Jamie Sharpe and Richard Self (eds.). Computers for Everyone. Derby: University of Derby, 2014.

[7] DYMEK, M. Communities build up Steam. **Pink Machine Papers**, v. 26, n. 5, p. 1-17, 2006.

[8] JOELSSON, T. N.; HYRYNSALMI, S.; MOLENAAR, S. The Role of Prosumers in the Evolution of a Software Ecosystem: Case Steam. In: Proceedings of the International Workshop on Software-intensive Business: Start-ups, Ecosystems and Platforms, 2018. CEUR-WS, 2018.

[9] THATIKONDA, V. K. Beyond the buzz: A journey through CI/CD principles and best practices. **European Journal of Theoretical and Applied Sciences**, v. 1, n. 5, p. 334-340, set. 2023.

[10] LIU, P.; SUN, X.; ZHAO, Y.; LIU, Y.; GRUNDY, J.; LI, L. A first look at CI/CD adoptions in open-source android apps. In: Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, 2022. IEEE/ACM, 2022.

[11] SHNEIDERMAN, B. **Designing the User Interface - 4th Edition**. Boston: Addison Wesley, 2004.

[12] AL-RUBAYE, H. A. T. **Towards the Automation of Migration and Safety of Third-Party Libraries**. 2020. Tese de Doutorado: Rochester Institute of Technology – Nova York, Estados Unidos. 2020.

[13] Kajima, D.; Kikuchi, H. **Failure of Privacy Policy for Session Replay Services Used for Monitor Your Keystroke**. In: International Conference on Network-Based Information Systems. Advances in Networked-based Information Systems. Springer Nature Switzerland, 2023.

[14] GRACIANO, L. P.; KAKINOHANA, G. K. **Avaliação de soluções de CI/CD: Uma Análise Comparativa entre GitHub Actions e GitLab CI**. 2023. Trabalho de Graduação: Universidade Federal de Mato Grosso do Sul – Campo Grande/MS, 2023.

[15] JOSHI, A.; KALE, S.; CHANDEL, S.; PAL, D. K. Likert Scale: Explored and Explained. **British Journal of Applied Science & Technology**, v. 7, n. 4, p. 396-403, 10 jan. 2015.

[16] LEHMAN, M. M. Programs, life cycles, and laws of software evolution. **Proceedings of the IEEE**, v. 68, n. 9, p. 1060-1076, set. 1980.