UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA - CCET
DEPARTAMENTO DE COMPUTAÇÃO - DC

Leandro Keller Salto

Desenvolvimento de uma Ferramenta de Apoio ao *Framework xPloiT* em Testes Exploratórios

SÃO CARLOS - SP
2025

Leandro Keller Salto

Desenvolvimento de uma Ferramenta de Apoio ao *Framework xPloiT* em Testes Exploratórios

Trabalho de conclusão de curso apresentado ao Departamento de Computação da Universidade Federal de São Carlos, para obtenção do título de bacharel em Ciência de Computação.

Orientador: Prof. Dr. André Takeshi Endo

SÃO CARLOS - SP
2025

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia - CCET
Departamento de Computação

Comissão avaliadora

Membros da comissão examinadora que avaliou e aprovou a Defesa do Trabalho de Conclusão de Curso do candidato Leandro Keller Salto, realizada em 11/12/2025

Prof. Dr. André Takeshi Endo
Instituição: Universidade Federal de São Carlos

Me. Yohan Duarte Pessanha
Instituição: Universidade Federal de São Carlos

Prof. Dr. Delano Medeiros Beder
Instituição: Universidade Federal de São Carlos

AGRADECIMENTO

Gostaria de expressar minha profunda gratidão aos meus pais, Fabio e Camila, que sempre me apoiaram em todas as decisões que tomei na vida, independentemente de concordarem com ela ou não.

Também expresso minha profunda gratidão ao meu orientador de TCC, Prof. Dr. André Takeshi Endo, pela oportunidade, e seu orientando Yohan Duarte Pessanha, por ser extremamente prestativo e sempre se voluntariar a me auxiliar, proporcionando a ajuda e o suporte necessários para a conclusão deste trabalho.

"Se alguém possuir tais qualidades, estudar o assunto por anos e dedicar toda a vida com perseverança invencível, então alcançará êxito, e poderá realizar melhoramentos extraordinários." — Charles Darwin, A Origem das Espécies (1859)

RESUMO

Este trabalho demonstra a reestruturação do Spring Espresso, um Software Web criado com o intuito de auxiliar a aplicação do *framework* $xPloiT$ em sessões de TE no contexto de jogos. Partindo de um protótipo desenvolvido por alunos da disciplina de Desenvolvimento de Software Web 1, na Universidade Federal De São Carlos, esta aplicação foi criada com o intuito de servir como ferramenta de apoio ao $xPloiT$ - um *framework* para realização de testes exploratórios em jogos. A principal evolução em relação ao protótipo inicial ocorreu na reestruturação do modelo de classes, que foi reorganizado a fim de oferecer uma representação mais consistente e alinhada em relação ao *framework*.

Esta reestruturação possibilitou a integração das entidades de forma coerente, resultando em uma base sólida para sustentar funcionalidades futuras, embora não esteja livre de limitações. O processo de refatoração revelou desafios significativos, como impactos em cascata ao modificar a relação e estrutura de classes previamente estabelecida, sendo necessário reiniciar o desenvolvimento para garantir robustez. Conclui-se que esta nova versão se encontra melhor estruturada para servir de apoio à aplicação da metodologia $xPloiT$.

Palavras-chave: Teste Exploratório, Engenharia de Software, Spring Boot, Modelagem de domínio.

ABSTRACT

This work demonstrates the restructuring of Spring Espresso, a web software created with the intention of assisting the application of the xPloiT framework in exploratory testing sessions in the context of games. Starting from a prototype developed by students of the Web Software Development 1 grade at the Federal University of São Carlos, this application was created with the intention of serving as a support tool for *xPloiT* - a framework for conducting exploratory testing in games. The main evolution in relation to the initial prototype occurred in the restructuring of the class model, which was reorganized in order to offer a more consistent and aligned representation in relation to the framework. This restructuring enabled the integration of entities in a coherent way, resulting in a solid base to support future functionalities, although it is not free from limitations. The refactoring process revealed significant challenges, such as cascading impacts when modifying the previously established class relationship and structure, making it necessary to restart development to ensure robustness. It is concluded that this new version is better structured to support the application of the xPloiT methodology.

Keyword: Exploratory testing, Software Engineering, Spring Boot, Domain modeling.

Lista de Figuras

Conteúdo

1 INTRODUÇÃO

O teste de software é uma parte importante para assegurar a qualidade, usabilidade e confiabilidade de aplicações. Conforme citado por Bach (2003), entre as abordagens existentes, o Teste Exploratório (TE) se destaca por ser dinâmico e investigativo, permitindo que o testador utilize sua experiência e conhecimento prévio para realizá-lo. Segundo Itkonen (2011), ao contrário dos testes roteirizados, esta técnica não depende de casos de teste definidos com antecedência, o que o torna mais útil em contextos de imprevisibilidade e mudanças frequentes, como do desenvolvimento de jogos digitais (Duarte, Politowski e Endo, 2025).

Nesse contexto, aplicações como jogos digitais apresentam particularidades que dificultam a criação de casos de teste previamente definidos, tais como interatividade constante, múltiplos caminhos possíveis e comportamentos não determinísticos (Duarte, Politowski e Endo, 2025). Para lidar com essas características, o TE em sessões surge como uma alternativa adequada, permitindo conduzir a atividade de forma estruturada sem comprometer a flexibilidade inerente à abordagem. Conforme apresentado em Lyndsay e Eeden (2003), essa técnica organiza o TE por meio de sessões com objetivos e limites de tempo previamente estabelecidos, equilibrando exploração e rastreabilidade. Dessa forma, o Teste Baseado em Sessões (TBS) fornece uma base consistente para o registro de evidências, observações e defeitos encontrados, se tornando particularmente relevante no contexto de jogos digitais.

Segundo Whittaker (2009), a exploração de software pode ser compreendida por meio da metáfora dos *tours*, que representam diferentes opções e abordagens para a condução dos testes. Na concepção do *framework xPloit* (Duarte, Politowski e Endo, 2025), esses *tours* foram adaptados para o contexto de jogos digitais e passaram a ser representados como estratégias de teste, como realizar uma sessão sob a ótica de um jogador inexperiente, de um jogador que domina e otimiza as mecânicas, ou de um jogador motivado pela obtenção de conquistas. Essa abordagem busca ampliar a cobertura dos testes, uma vez que perspectivas distintas tendem a conduzir o jogador por caminhos ainda inexplorados, possibilitando a identificação de situações que poderiam passar despercebidas em abordagens tradicionais.

Baseando-se na necessidade de adaptar o TE para o contexto de jogos digitais, o *framework xPloiT*, apresentado em Duarte, Politowski e Endo (2025) surge como uma abordagem estruturada composta por nove estratégias específicas para o domínio, visando orientar o testador, definindo uma ordem recomendada de aplicação de estratégias, a fim de auxiliar na detecção de *bugs* e falhas inesperadas, tornando assim o TE mais sistemático para realização de testes manuais em jogos, sem restringir a característica investigativa da técnica.

Com o intuito de apoiar a aplicação do *xPloiT*, este estudo apresenta o Spring Es-

presso, um sistema que atende às demandas do *framework*, proporcionando artifícios para documentação e organização das sessões de teste. Embora voltada primariamente para o domínio de jogos, o sistema pode ser estendido para outras áreas com características semelhantes, como múltiplos caminhos de interação, dinamicidade e necessidade de feedback rápido. Desta forma, o objetivo deste estudo é demonstrar a evolução do protótipo original para uma versão mais completa e aderente às diretrizes do *xPloit*.

1.1 Objetivos

Este trabalho visa apresentar o desenvolvimento da ferramenta Spring Espresso, um software web implementado para auxiliar na aplicação do *framework xPloiT*. A ferramenta padroniza e facilita o registro de sessões, documentação de *bugs* e gerenciamento de projetos por parte das equipes, com uma interface gamificada, interativa e simples. Desta forma, há a diminuição da informalidade e a perda de rastreio do progresso, frequentemente associadas ao TE, promovendo maior eficiência para o processo de teste.

Para isso, o sistema permite a criação de usuários com diferentes níveis de acesso (sendo eles testador ou administrador), com a funcionalidade de criação de projetos e alocação de usuários para realização dos testes. As sessões de teste criadas estarão vinculadas à uma estratégia específica, baseado nas metáforas dos *tours*. Em cada uma delas, os testadores poderão registrar os *bugs*, dentro de um tempo limite previamente estipulado na hora da criação.

1.2 Estrutura do Trabalho

Este trabalho está dividido da seguinte forma: o Capítulo 2 discute a revisão bibliográfica, abordando conceitos fundamentais para o entendimento do estudo e trabalhos e ferramentas relacionadas; o Capítulo 3 detalha a metodologia e material utilizado para a condução do estudo; no Capítulo 4 são apresentados e analisados os resultados obtidos; e, no Capítulo 5, são apresentadas as conclusões e sugestões para trabalhos futuros.

2 REVISÃO BIBLIOGRÁFICA

Este capítulo apresenta conceitos importantes para o entendimento do estudo, trabalhos e ferramentas relacionadas, que sustentam o desenvolvimento deste projeto. Inicialmente, na subseção 1, se discute o conceito de TE, suas principais abordagens e técnicas sistematizadas, o gerenciamento de sessões. Em seguida, na subseção 2, são exploradas pesquisas e ferramentas que aplicam ou ampliam o TE em diferentes contextos, incluindo jogos digitais, abordando desde propostas de treinamento e gamificação até *frameworks* e soluções de apoio ao processo exploratório.

2.1 Teste Exploratório Em Jogos Digitais

Embora frequentemente seja percebido como uma atividade aleatória ou desestruturada, o TE é descrito como um ciclo contínuo de aprendizado, projeto e execução (Castro, 2018). Essa visão é defendida por Bach (2003), que relata eventos práticos da vida, com anedotas, concluindo que o TE pode ser mais produtivo do que testes roteirizados em diversas ocasiões, sobretudo quando combinado com testes em sessões[1]. Cada sessão possui início e fim definidos com um objetivo específico de exploração, permitindo ao testador investigar partes limitadas do sistema com liberdade, mas dentro de um escopo controlado. Ao final, o testador registra suas ações, observações e resultados em um relatório, o que possibilita mensurar cobertura, risco e progresso, mantendo a flexibilidade e o aprendizado característicos do TE, que garantem melhor rastreabilidade e mensurabilidade do que está sendo testado, ideal para contextos altamente dinâmicos, como por exemplo, jogos.

Uma das contribuições mais conhecidas para tornar o TE mais sistematizado é a proposta dos *tours* de Whittaker (2009), que introduzem um conjunto de metáforas de exploração inspiradas em passeios turísticos para orientar o testador durante a investigação do sistema. Cada *tour* sugere uma forma específica de percorrer o software, direcionando a atenção do testador para diferentes funcionalidades, fluxos, exceções e comportamentos do sistema. Ao empregar esse vocabulário comum, os *tours* auxiliam na organização do raciocínio exploratório, reduzindo a aleatoriedade da atividade e incentivando uma exploração consciente. Dessa forma, a abordagem contribui para que o TE seja conduzido de maneira mais estruturada, mantendo sua natureza flexível, ao mesmo tempo em que fornece diretrizes práticas para a definição de objetivos e a variação sistemática das estratégias de teste.

Lyndsay e Eeden (2003) propuseram o gerenciamento de TBS como uma abordagem para organizar a atividade de TE por meio de unidades de trabalho com duração previamente definida e propósito explícito. Cada sessão estabelece um foco específico de

[1]Os testes em sessões são uma técnica para gerenciar e controlar testes não roteirizados, estruturando-os em períodos de tempo delimitados denominados *sessões*.

investigação, ao mesmo tempo em que preserva a autonomia do testador para conduzir a exploração conforme as descobertas realizadas. O estudo apresenta evidências empíricas obtidas em projetos reais, indicando que a adoção dessa técnica contribui para o aumento da rastreabilidade, da comunicação dos resultados e do controle sobre o progresso dos testes. Dessa forma, o TBS se mostra uma alternativa robusta para apoiar a realização de testes exploratórios em contextos de diferentes escalas, introduzindo disciplina e visibilidade a uma atividade que tradicionalmente ocorre de maneira menos estruturada.

O estudo de Duarte et al. (2024) investiga estratégias e desafios da abordagem, aplicando experimentalmente sessões de TE em cinco jogos de plataforma (tanto 2D quanto 3D) para avaliar jogabilidade, ocorrência de *bugs*[2] e percepções de jogo. Os autores adaptaram sete estratégias de Whittaker, definiram um processo adaptado ao domínio e conduziram as sessões de teste conforme esse procedimento. Todas as sessões foram gravadas e executadas por dois dos autores, os *bugs* encontrados foram registrados em planilha, classificados por severidade e analisados, o que permitiu extrair percepções sobre a eficácia das estratégias de TE em jogos de plataforma.

Em continuidade a esse trabalho, Duarte, Politowski e Endo (2025) propõem o *framework xPloiT*, que formaliza e amplia a aplicação do TE no contexto de jogos de plataforma. O *framework* organiza a atividade de teste em nove estratégias exploratórias sequenciais, derivadas dos *tours* de Whittaker, as quais representam diferentes perspectivas de interação com o jogo e orientam o raciocínio do testador durante as sessões. As estratégias são:

- Newbie Journey, que consiste em deixar o jogador ter um primeiro contato com o jogo, jogando da forma que jogaria normalmente.
- Golden Path, que visa apresentar ao jogador a maneira ótima de se jogar, ou seja, seguir o caminho recomendado pelos desenvolvedores, de maneira otimizada e direta, evitando desvios desnecessários, e rotas indesejadas.
- Noob Journey, que incentiva o jogador a jogar o jogo de maneira intencionalmente incorreta, não seguindo recomendações, e buscando maneiras alternativas de realizar as ações.
- Completionist, que consiste em jogar o jogo de maneira a realizar todas as ações possíveis dentro dele, como pegar todas as moedas, eliminar todos os inimigos, abrir todas as portas, etc, de maneira que não reste nenhum objetivo incompleto na fase.
- Stress Test, que incentiva o testador a realizar comandos inválidos, ou não esperados pelos desenvolvedores, com o intuito de provocar reações inesperadas no jogo.

[2]Em testes de software, o termo *bug* designa um erro, falha ou defeito no código que provoca comportamentos inesperados, comprometendo a jogabilidade, a estabilidade ou a experiência do usuário (Widder e Goues, 2024).

- Speedrun, que instrui o testador a finalizar a fase da maneira mais rápida possível, fazendo o mínimo de ações necessárias para concluir o objetivo.
- User Interface, que busca testar menus, interfaces e gráficos do jogo, de maneira a testar se estão funcionando da maneira adequada.
- Neighboring, onde o usuário tenta encontrar *bugs* realizando ações e em locais e circunstâncias nas quais outros *bugs* já foram encontrados préviamente.
- Overtime, que busca testar novamente *bugs* conhecidos, em um outro momento, aplicando o "neighboring" após melhorias e evoluções.

Essas estratégias resultam da adaptação de técnicas clássicas de TE para o domínio de jogos digitais, incorporando também conceitos do *game design* e de perfis de jogadores, como *golden path*, *completionist* e *speedrun*. Dessa forma, o *xPloiT* fornece um guia adaptável que sistematiza o processo exploratório sem restringir sua natureza criativa, permitindo registrar, comparar e analisar sessões de teste de maneira mais rastreável e mensurável. Assim, a proposta representa um avanço teórico e prático ao alinhar métodos consolidados de TE às particularidades dos jogos digitais, equilibrando liberdade investigativa e controle metodológico.

2.2 Trabalhos Relacionados

No contexto de TE aplicado em jogos digitais, Silva et al. (2024) mostraram que a preocupação com o uso de boas práticas de experiência de usuários (do inglês *User Experience*, ou UX) podem aprimorar a etapa de testes. O estudo contou com 13 desenvolvedores de jogos de diferentes regiões e tamanhos de empresa e revelou que mesmo aplicações informais de UX ajudam a reduzir frustrações e melhorar a coleta de *feedback* dos jogadores, ampliando a qualidade das interações.

Micallef, Porter e Borg (2016) demonstraram que o nível de capacitação das equipes é um elemento determinante para a eficácia do TE. No artigo, compararam-se grupos com e sem treinamento formal, observando-se que cada um identificou tipos distintos de falhas. Em vez de uma relação hierárquica entre os resultados, constatou-se uma complementaridade entre as descobertas, indicando que equipes heterogêneas, formadas por testadores com diferentes níveis de experiência e abordagens, tendem a obter uma cobertura mais ampla e eficaz durante as sessões de TE.

Tratando-se do impacto da gamificação no contexto de TEs de interfaces gráficas, Coppola et al. (2023) realizaram um experimento que envolveu sessenta participantes divididos em grupos, comparando o desempenho em cenários com e sem elementos de jogo. Os resultados indicaram que a gamificação aumentou o engajamento dos testadores, reduziu o tempo necessário para identificar falhas e proporcionou uma experiência mais

motivadora durante a execução dos testes. Entretanto, os autores ressaltam a importância de equilibrar o nível de semelhança com jogos tradicionais, a fim de evitar que os elementos lúdicos se tornem uma distração e comprometam a atenção do testador. Essa abordagem destaca que a motivação e a imersão do testador são aspectos essenciais para a eficácia do TE em sistemas interativos.

Pesquisadores têm conduzido comparações sistemáticas entre diferentes estratégias de teste para evidenciar as particularidades de cada abordagem (roteirizadas ou exploratórias). Neste contexto, Itkonen (2011) analisa diversos estudos empíricos sobre o uso do TE, concentrando-se na comparação entre abordagens exploratórias e roteirizadas, em termos de eficácia e eficiência. As análises indicam que o TE pode alcançar níveis de efetividade semelhantes ou superiores aos testes baseados em casos, especialmente pela capacidade do testador de aplicar seu conhecimento e experiência para identificar comportamentos anômalos e falhas sutis. Apesar disso, o TE apresenta desafios quanto à repetibilidade e rastreabilidade, e seus resultados dependem fortemente do conhecimento técnico do testador. A principal contribuição da revisão é a sistematização de evidências empíricas que reforçam a legitimidade científica da prática.

No ambiente ágil, integrar o TE às práticas de desenvolvimento demanda abordagens que conciliem flexibilidade e rastreabilidade. Coutinho, Andrade e Machado (2023) propõem o Agile ETeasy, desenvolvido com base na metodologia de *Design Science Research*, método de pesquisa orientado ao problema. No artigo, os autores demonstram que esta nova proposta organiza o TE em etapas de planejamento, execução e encerramento, com técnicas colaborativas e artefatos simples, favorecendo sua aplicação em ciclos curtos de *sprints* e em contextos de mudanças frequentes de requisitos.

Além das abordagens metodológicas, diversas ferramentas de apoio ao teste buscam aprimorar o registro, rastreamento e a execução de testes, com diferentes níveis de suporte ao TE. Entre elas, o *Xray*[3], amplamente utilizado em conjunto com o Jira para o gerenciamento de requisitos, planos e evidências de teste. A ferramenta permite gerenciar casos de teste manuais e automatizados como itens do Jira, oferecendo rastreabilidade completa entre requisitos, defeitos e execuções. Contudo, seu foco principal está na rastreabilidade e no controle de planos de teste pré-definidos, o que pode limitar a flexibilidade necessária em atividades exploratórias.

Uma extensão de navegador desenvolvida por Leveau et al. (2022) monitora cliques e interações, usando modelos de predições baseados em *n-gramas*.[4] Os *n-gramas* representam sequências de interações do testador com o sistema, permitindo ao modelo sugerir caminhos ainda não alcançados e ampliar a diversidade de exploração para indicar caminhos pouco explorados. Em testes com sessenta participantes, a ferramenta levou a uma

[3]Disponível em: <https://marketplace.atlassian.com/apps/1211769/xray-test-management-for-jira> . (28 de Agosto de 2025)

[4]Um *n-grama* é uma sequência de n elementos consecutivos, como palavras ou caracteres — usada para identificar padrões e prever ocorrências futuras (Cavnar, Trenkle et al., 1994).

maior variedade de ações e, consequentemente, a mais descobertas de falhas, justamente pelo fato do uso de *n-gramas* possibilitar uma melhor cobertura de exploração.

Outras soluções, como o *TestRail*[5], seguem lógica semelhante: priorizam a documentação e o controle de execução de testes roteirizados, ainda que incluam suporte básico a sessões exploratórias. No entanto, esse suporte se restringe ao registro descritivo de observações, sem oferecer mecanismos específicos para orientar ou analisar o processo exploratório em si.

O *PractiTest*[6], por sua vez, apresenta maior flexibilidade ao permitir a criação e execução de testes não roteirizados, integrando-se tanto a fluxos manuais quanto automatizados. A ferramenta possibilita o registro de sessões exploratórias e o acompanhamento de resultados em tempo real.

Em contraste, a solução proposta neste trabalho, que tem como base princípios do *framework xPloiT*, foi desenvolvida para integrar o registro de evidências com a análise das rotas exploradas, priorizando o contexto do domínio de jogos digitais. Diferentemente das ferramentas mencionadas, o *xPloiT* não parte de casos de teste predefinidos, mas de metas de exploração adaptáveis, onde os envolvidos nos testes podem criar sessões para cada estratégia que será utilizada, a fim de se obter um melhor direcionamento para a realização dos TEs, permitindo acompanhar o fluxo de descobertas em tempo real e fortalecer tanto a rastreabilidade quanto o aprendizado durante o processo exploratório.

[5]Disponível em: <https://www.testrail.com/agile-testing> (28 de Agosto de 2025).
[6]Disponível em: <https://www.practitest.com> (28 de Agosto de 2025).

3 METODOLOGIA DO ESTUDO

Este capítulo tem como objetivo descrever o processo metodológico utilizado para o desenvolvimento da versão final do Spring Espresso, desde as etapas iniciais de planejamento e compreensão do projeto, até a análise do protótipo existente, identificando limitações, oportunidades de melhorias e adaptações desejáveis para atingir o objetivo de torná-lo uma ferramenta de apoio voltada à aplicação da *framework xPloiT*. Todas as análises foram baseadas em estudos teóricos, avaliações técnicas, discussões com os pesquisadores envolvidos e experimentação prática, garantindo um processo evolutivo sustentável para a ferramenta.

3.1 Planejamento inicial do estudo

Inicialmente, este estudo definiu o escopo da ferramenta: uma aplicação que propicie e facilite o processo de TEs, principalmente no contexto de jogos digitais, usando conceitos de sessões cronometradas, estratégias de exploração e uma forma eficiente de registro de *bugs*. Nas reuniões semanais, estabeleceram-se objetivos centrais da pesquisa, tais como os método de trabalho, partindo-se de um protótipo pré-existente da aplicação, sendo necessário analisar a estrutura do código, identificar limitações e propor aprimoramentos desejáveis.

Esta etapa foi crucial para o desenvolvimento da ferramenta, estabelecendo um objetivo claro, assim como a definição de metas, propiciando ao desenvolvimento um caminho a ser seguido, podendo ter seu sucesso mensurado.

3.2 Tecnologias Utilizadas

Para o desenvolvimento desta ferramenta, foram utilizadas algumas tecnologias complementares que atuam em diferentes camadas da aplicação, tanto para o *backend*, como *frontend*, colaborando para garantir organização e eficiência no desenvolvimento do sistema. São elas:

- ***Spring Boot***: um *framework* de código público apoiado em Java, com o objetivo de simplificar a criação de aplicações *Spring* por automatizações e convenções padronizadas (VMware, Inc., 2025). Isso permite que o programador se concentre na lógica das entidades e os seus relacionamentos, evitando a necessidade de configurações extensivas e repetitivas.

- ***Thymeleaf***: um mecanismo de template utilizado em aplicações web Java, que serve para gerar páginas dinâmicas em HTML de forma simples, diretamente integrada com o *Spring Boot* (Team, 2025). Ele permite a fácil construção de interfaces

interativas, mantendo o HTML válido e interpretável, tanto para o servidor quanto para o navegador.

- ***MySQL***: é um Sistema Gerenciador de Banco de Dados amplamente utilizado no desenvolvimento de software e aplicações web, o qual é eficiente, confiável e compatível com diversas linguagens de programação. Por se tratar de um banco de dados relacional, ele organiza as informações em tabelas que se relacionam por meio de chaves primárias e estrangeiras, o que possibilita uma consulta eficiente e consistente mesmo com grandes volumes de dados (Corporation, 2025).

- ***Spring Data JPA***: proporciona uma maior facilidade para se trabalhar com banco de dados no desenvolvimento de aplicações. Ele permite criar e buscar informações no banco de dados sem precisar escrever comandos *SQL*, tornando o desenvolvimento mais fluido e com menor ocorrência de erros (VMware, Inc., 2025).

- ***xPloiT***: é uma proposta criada a fim de organizar e orientar a execução de TEs no contexto de jogos digitais. Ele reúne ao todo nove estratégias, que quando aplicadas guiam o testador a adotar uma perspectiva diferente de exploração. O principal objetivo do *framework* é ampliar a descoberta de falhas de jogabilidade, bem como proporcionar uma abordagem estruturada e eficaz ao teste manual de jogos, agregando repetibilidade e consistência.

3.3 Análise do Protótipo e Definição do escopo

Para esta etapa, utilizou-se como base um protótipo previamente criado na matéria de Desenvolvimento de Software Web 1, disponível em repositório público no GitHub[7].

Nesta primeira versão, o sistema oferecia um conjunto inicial de funcionalidades típicas de um gerenciador de TEs, incluindo o cadastro de diferentes estratégias de teste, cada uma com características próprias. Cada estratégia continha atributos como "Exemplos", que apresentavam possíveis cenários de aplicação, e "Dicas", que forneciam orientações para auxiliar o testador a compreender e potencializar o uso daquela abordagem durante a execução das sessões.

A primeira tela disponível no protótipo é a de login, na qual o usuário escreve seu usuário e senha, e com os dados consolidados, poderá se conectar ao sistema, conforme mostrado na Figura 1.

Pensando nisso, os usuários cadastrados no sistema, poderiam ser alocados em "Projetos", vistos em Figura 2, nos quais realizariam sessões de testes com tempo pré-definido, indicando qual estratégia estaria sendo utilizada e registrando os *bugs* encontrados durante a execução, de maneira simples.

[7] <https://github.com/ronanpjr/SpringEspresso>

Figura 1: Tela de *Login* do protótipo (Fonte: Autoria própria).

Figura 2: Tela de listagem de Projetos do protótipo (Fonte: Autoria própria).

Figura 3: Tela de gerenciamento de usuários do protótipo (Fonte: Autoria própria).

Figura 4: Tela de edição de usuários do protótipo (Fonte: Autoria própria.

Além disso, o sistema apresenta diferentes cargos, com a possibilidade de se conectar como administrador, função na qual o usuário poderia gerenciar a lista de usuários, como apresentado nas Figuras 3 e 4, criar novos projetos, ilustrado na Figura 5 e adicionar novas estratégias para serem utilizadas nas sessões, visível nas Figuras 6 e .

Neste protótipo, a entidade "Projeto" pode ter várias entidades do tipo "Sessão", como ilustrado na Figura 8, que por sua vez, possuem uma única estratégia associada, visto na Figura 9. Esta estrutura é ideal para a realização de TEs, uma vez que como demonstrado por Lyndsay e Eeden (2003), sessões cronometradas e com um objetivo bem definido contribuem para um direcionamento eficiente da exploração no tempo da execução.

A fim de otimizar a interação do usuário, na tela de Sessão se encontram botões para o registro de *bugs* de maneira rápida, reduzindo o tempo gasto com ações de documentação, permitindo que o testador se concentre na execução do teste, ilustrado na Figura 10.

Para facilitar o processo de identificação, os *bugs* encontrados precisam apenas de uma descrição, podendo ser salvos, registrando a hora e data que foram encontrados, tendo seu

Figura 5: Tela de criação de projetos do protótipo (Fonte: Autoria própria).

Figura 6: Tela de listagem de estratégias do protótipo (Fonte: Autoria própria).

Figura 7: Tela de detalhes das estratégias do protótipo (Fonte: Autoria própria).

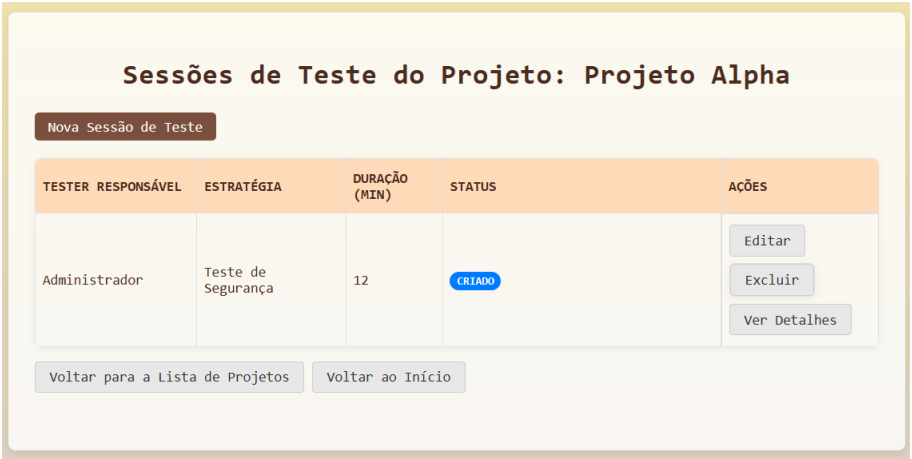

Figura 8: Tela de detalhes dos projetos do protótipo (Fonte: Autoria própria).

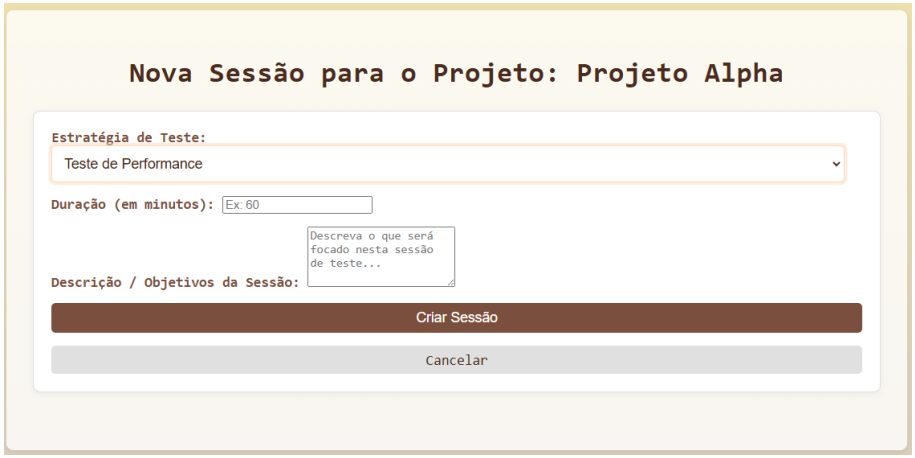

Figura 9: Tela de criação de sessões do protótipo (Fonte: Autoria própria).

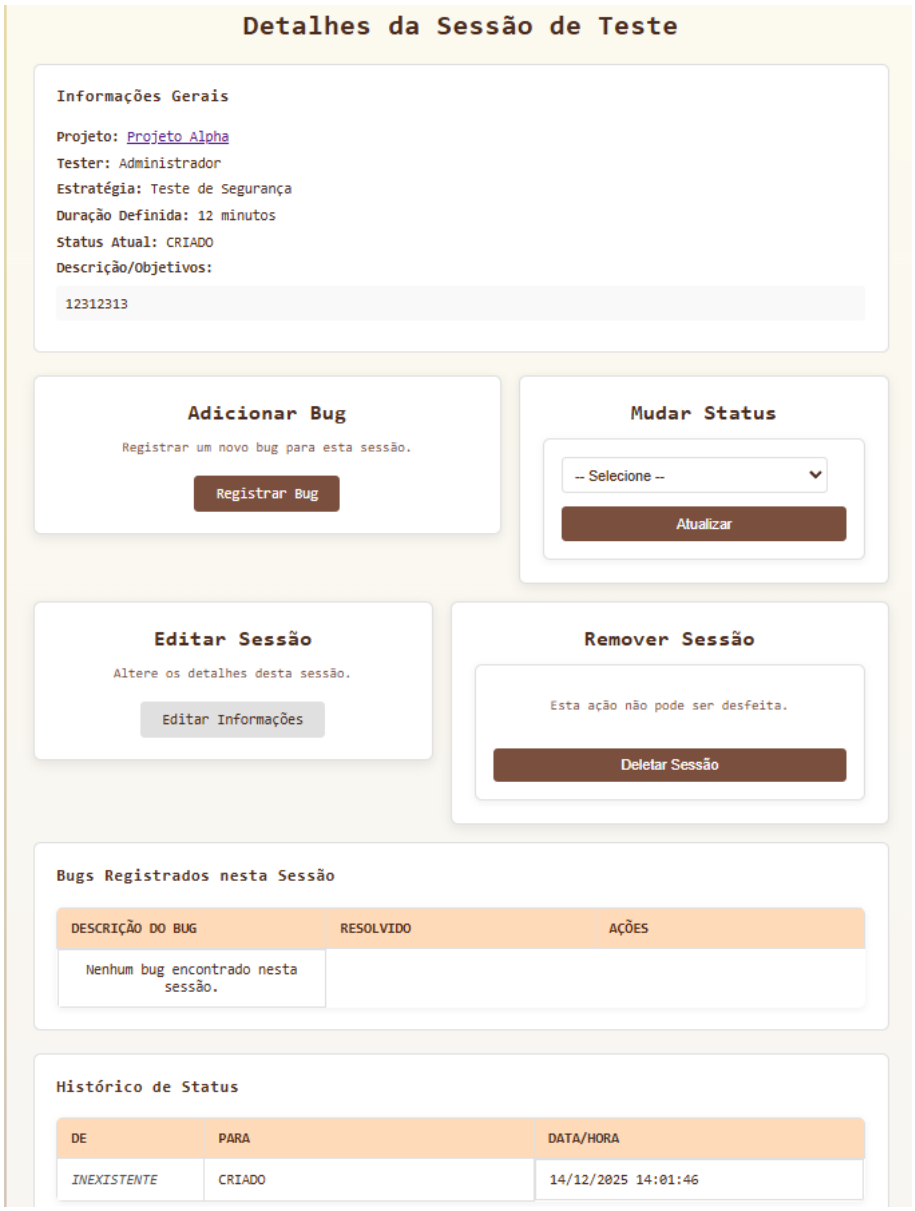

Figura 10: Tela de detalhes da sessão do protótipo (Fonte: Autoria própria).

status como “Aberto” por padrão, com a possibilidade de ser alterado para “Resolvido” posteriormente caso seja resolvido no futuro, disponível na Figura 12. A Figura 11 ilustra a estrutura de classes do protótipo.

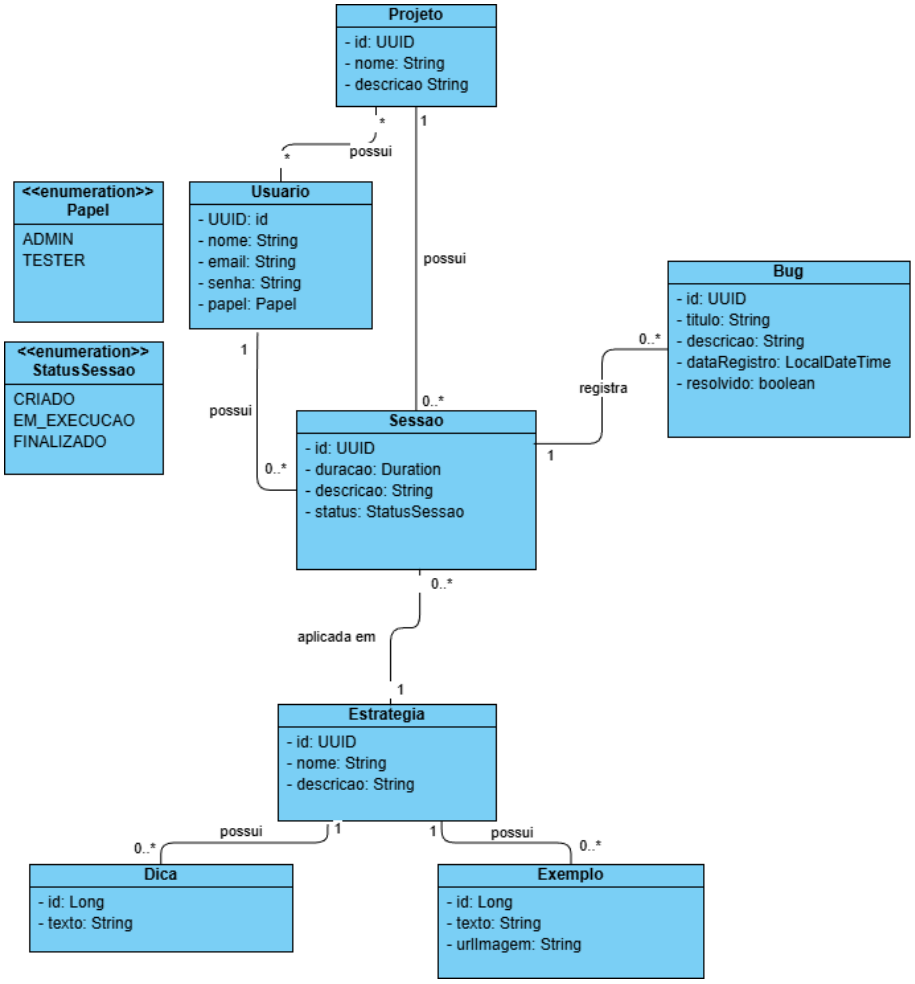

Figura 11: Estrutura de classes do protótipo (Fonte: Autoria própria).

Estas são as principais funcionalidades desta primeira versão do Spring Espresso, que se mostrou adequada para realizar sessões de TEs em diversos contextos e situações, com característica genérica e flexível para documentações em times que buscam uma forma ágil e eficiente de registrar e gerenciar suas sessões de teste.

3.4 Limitações do Protótipo

Apesar do protótipo inicial do Spring Espresso ser uma ferramenta útil na documentação e organização de testes exploratórios, ele ainda apresenta algumas lacunas fundamentais, principalmente para a execução do framework *xPloit*. São elas:

- Sessões de teste são criadas diretamente dentro de um projeto, podendo o usuário escolher a estratégia que deseja utilizar, reduzindo a cobertura das estratégias.

Figura 12: Registro de *bug* do protótipo (Fonte: Autoria própria)

- Carência de detalhes na documentação de *bugs*, pois não é possível fazer registros visuais dos *bugs*, como fotos ou vídeos. Isto limita o entendimento do usuário e reduz a capacidade de reprodução do *bug*.
- Não há um impedimento de registrar *bugs* sem que a sessão tenha se iniciado ou passado do seu tempo estipulado, o que pode acarretar em inconsistências e incoerências nas análises futuras.
- Não há uma tela contendo todos os *bugs* encontrados, reduzindo a visão dos usuários.
- Não existem elementos de gamificação nas telas do sistema. A falta deste elemento pode diminuir o engajamento dos testadores na realização dos testes.

4 EVOLUÇÃO DO SPRING ESPRESSO

Neste capítulo são analisadas as mudanças realizadas no Spring Espresso desde o seu protótipo inicial, apresentando a repaginação de telas, alteração de comportamento e melhorias gráficas, além disso são apresentadas mudanças conceituais do projeto. Também se discorre sobre a mudança de estrutura e modelagem das classes, a fim de evidenciar as melhorias técnicas do projeto, complementadas com reflexões e lições aprendidas.

4.1 Análise das mudanças

A nova versão, disponível em repositório público no GitHub[8] representa uma significativa evolução em relação ao protótipo inicial. Ao todo, foram 21 *commits* desde o momento da criação do *fork* no GitHub, até a finalização da aplicação, com a adição de 9295 linhas, e a deleção de 7157, como ilustrado na Figura 13, que resultou num aumento de 2138 novas linhas de código.

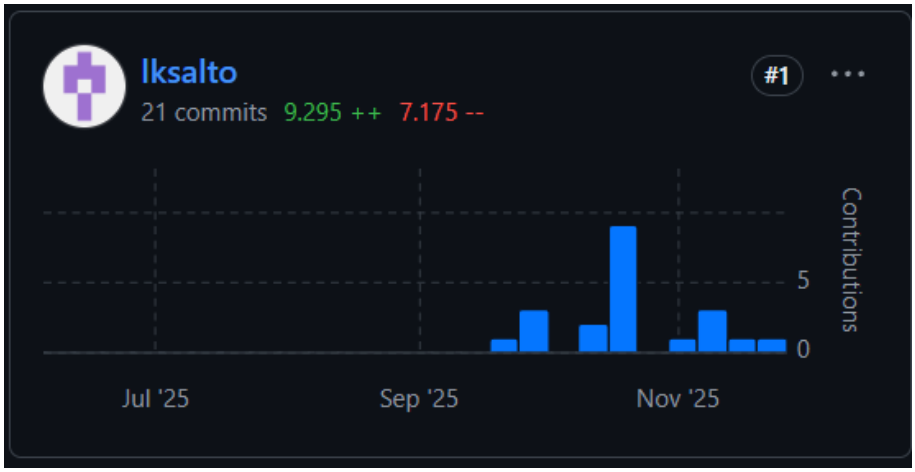

Figura 13: Contagem de commits e número de linhas adicionadas e removidas (Fonte: Autoria própria).

A cada *commit*, o projeto passava por melhorias estruturais e conceituais tanto na modelagem das entidades, quanto no funcionamento do sistema. Tais mudanças buscam aprimorar a realização de testes no contexto da utilização do framework *xPloiT*, alinhado com fundamentos teóricos sólidos e adequados ao uso em contextos de TEs para jogos eletrônicos.

Entre as melhorias implementadas, destacam-se o *redesign* das telas principais, com uma disposição mais clara dos elementos, melhor aproveitamento do espaço e aumento do tamanho das tabelas e campos de texto, tornando a navegação e leitura mais confortáveis. Foram ocultados campos técnicos considerados desnecessários na interface, como identificadores (`id`), que antes sobrecarregavam a visualização.

[8] <https://github.com/lksalto/SpringEspresso>

Também foram corrigidos erros relacionados ao registro de entidades, incluindo falhas no *upload* e persistência de imagens, as quais não estavam sendo apresentadas corretamente na interface do sistema, gerando uma impressão de que o arquivo não foi propriamente salvo. Adicionou-se a possibilidade de anexo no registro de *bugs*, podendo o usuário escolher entre salvar uma foto ou um curto vídeo do *bug*, melhorando o entendimento do contexto em que ele foi detectado.

Essas mudanças visam não apenas corrigir problemas técnicos, mas também aprimorar a fluidez da interação com o sistema, tornando o uso mais intuitivo e alinhado às necessidades reais dos projetos, proporcionando um processo de registro, documentação e análise de sessões de TE mais fiel às diretrizes do *framework* e facilitando as ações dos testadores.

4.1.1 Tela de *login*

A fim de manter o acesso ao Spring Espresso controlado, é necessário passar por um processo de cadastro de novos usuários, realizado por meio da tela de registro de usuários, que pode ser utilizada apenas por administradores do sistema. Porém, é possível acessar as estratégias cadastradas como visitante, sem a necessidade do registro. Para se conectar basta fornecer um e-mail e senha válidos. A Figura 14 apresenta a tela inicial do sistema, com campos para o usuário adicionar seu e-mail e senha para se conectar, ou entrar como visitante.

Figura 14: Tela de *Login* (Fonte: Autoria própria).

4.1.2 Tela Inicial

Na tela inicial do Spring Espresso existem quatro botões que levam à áreas distintas do programa, ilustrados na Figura 15. A primeira faz referência à tela de gerenciamento de usuários, funcionalidade restrita à administradores do sistema, onde é possível adicionar, remover e alterar dados das pessoas cadastradas. Há tambem o botão de gerenciamento de projetos, o qual pode ser acessado por todos os usuários cadastrados e apresenta os projetos existentes no sistema. O botão de explorar estratégias pode ser acessado mesmo

por pessoas que não estão cadastradas, e nela estão contidas informações relevantes sobre as estratégias cadastradas no sistema, ação que só pode ser realizada por administradores. Por último, há o botão para acessar *bugs* encontrados, que apresenta uma tabela com os registros de *bugs* de todos os projetos, fornecendo uma visualização do andamento dos testes, conforme ilustrado na Figura 16.

Figura 15: Tela inicial (Fonte: Autoria própria).

4.1.3 Gerenciamento de Usuários

Como citado, esta tela é de acesso único de administradores, que podem adicionar, remover e editar outros usuários.

Sua principal função é garantir que as entidades Usuário possuam uma forma simples e eficiente de gerenciamento, na qual o administrador possa realizar sua atividade com poucos cliques. De acordo com o que ilustra a Figura 16, nesta tela estão contidos um botão para adicionar novos usuários e uma tabela com uma listagem de todas as contas registrados. Há também um botão para retornar à pagina inicial.

Na última coluna da tabela há três botões de ação, que permitem ver detalhes do usuário, editá-lo ou removê-lo do sistema. O botão de novo usuário levará à tela de registro novamente.

Para o cadastro desta entidade, é necessário que o administrador forneça o nome, e-mail, senha e cargo, com a possibilidade de alteração futura, caso necessário. A senha possui uma restrição de mais de seis caracteres, para adicionar complexidade e segurança, ao passo que o e-mail precisa conter o "@" e mais um caractere posteriormente, a fim de diminuir a ocorrência de inconsistências, como mostrado na Figura 17.

Figura 16: Tela de listagem dos usuários registrados no sistema (Fonte: Autoria própria).

Figura 17: Tela de cadastro de usuários (Fonte: Autoria própria).

4.1.4 Gerenciamento de Projetos

Na primeira versão, a entidade Sessão possuía o papel de destaque, representando o principal elemento no contexto dos testes. Cada uma delas estava associada a um testador, um projeto e uma única estratégia.

Após a atualização, o foco do teste foi deslocado para a entidade Estratégia, que passou a ser o grande eixo central do sistema. Esta decisão foi fundamentada nos trabalhos

de Whittaker (2009) e Duarte, Politowski e Endo (2025), mesclando a metáfora dos *tours* com aplicações práticas de TEs no contexto de jogos. Sendo assim, as estratégias implementadas no Spring Espresso servem como pilares para guiar os testadores a fim de terem uma melhor organização, direcionamento, eficiência e documentação.

Neste sentido, a entidade Estratégia deixa de ser apenas um atributo da entidade Sessão para se tornar o ponto focal do ciclo de teste. Cada projeto, ao ser criado, terá suas estratégias pré-definidas pelo administrador, que terá total liberdade para adicioná-las no momento de sua criação, como ilustrado na Figura 18.

Ainda na criação do projeto, o administrador deverá selecionar seus membros, ou seja, os testadores que terão acesso a criar sessões para as estratégias, onde irão documentar os *bugs* encontrados e contribuir para os testes realizados. A modificação dos participantes pode ocorrer de maneira livre, no entanto, de modo a manter a consistência da metodologia aplicada no projeto, a coleção de estratégias alocadas não pode ser alterada após definida, conforme ilustrado na Figura 19.

Após a criação, o administrador é redirecionado para a tela de listagem de projetos, visível na Figura 20, onde os participantes podem acessar os detalhes de cada projeto. A partir dessa visualização, é possível realizar ações como criar novas sessões ou removê-las.

Figura 18: Tela de cadastro de novo projeto (Fonte: Autoria própria).

Ao clicar em Ver Detalhes, representado na Figura 20 o usuário é redirecionado à tela com mais informações sobre o projeto, que apresenta as estratégias alocadas, cada

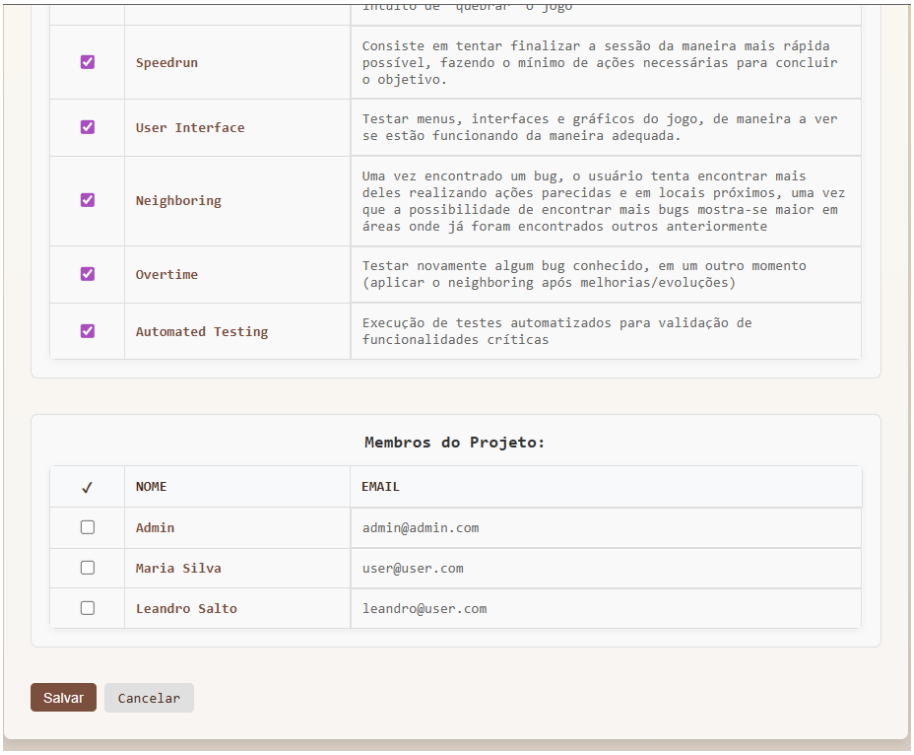

Figura 19: Campo de adição de testadores para um projeto (Fonte: Autoria própria).

Figura 20: Tela de projetos pela visão do testador (Fonte: Autoria própria).

uma com um indicador visual de como está o andamento das sessões. Nesta tela, estão aplicados elementos de gamificação e teorias de *UX/UI* (Ver Capítulo 2) para uma melhor experiência do usuário, como a representação visual de progresso:

- Estratégias sem sessões finalizadas, não possuem medalhas;
- Estratégias com pelo menos uma sessão finalizada recebem uma medalha de bronze;
- Estratégias com cinco a nove sessões finalizadas recebem uma medalha de prata;
- Estratégias com dez ou mais sessões finalizadas recebem uma medalha de ouro;

- Os botões da interface também possuem cores correspondentes às medalhas, oferecendo um *feedback* visual imediato e facilitando a interpretação das informações pelo usuário.

A Figura 21 traz detalhes sobre um projeto específico, apresentando quais estratégias estão alocadas, bem como o andamento das sessões representadas graficamente conforme discutido no parágrafo anterior.

Figura 21: Tela de detalhes de um projeto (Fonte: Autoria própria).

Na nova versão, as sessões não são mais instâncias isoladas dentro do projeto, mas sim derivadas diretamente das estratégias associadas, o que significa que ao iniciar uma nova sessão, o testador estará enriquecendo aquele projeto com mais uma aplicação da estratégia, ou seja, o *xPloiT* estará sendo aplicado.

Clicando em um dos botões das estratégias, o usuário terá acesso à lista de sessões realizadas para aquela estratégia específica, onde haverá a descrição da estratégia escolhida, bem como uma tabela com informações sobre as sessões já existentes, evidenciado na Figura 22.

Se o usuário clicar no botão Nova Sessão, será redirecionado para a tela de cadastro de sessões, na qual terá de informar uma descrição do que será testado, bem como sua duração estipulada. Caso seja um administrador, ele pode também selecionar quem será o testador da sessão. Uma vez criada, a ferramenta volta para a tela de listagem das sessões, conforme a Figura 23.

Para iniciar uma sessão de teste, o usuário deve clicar no botão Detalhes, apresentado na Figura 22, que o levará à página de detalhes da sessão, tela na qual será possível ver informações como duração, status e descrição. Toda sessão se inicia com o *status* de criada, não podendo ter *bugs* registrados, pois ainda não foi iniciada.

Figura 22: Tela de sessões de teste de uma estratégia (Fonte: Autoria própria).

Figura 23: Tela de criação de sessões, do ponto de vista de um testador (Fonte: Autoria própria).

Enquanto a sessão não tiver seu status atualizado para Em Execução, não será possível registrar *bugs*; isto garante consistência dos dados, sincronizando tempo de execução de sessões com horário em que *bugs* foram encontrados. Esta proteção contra inconsistências também se aplica à sessões já finalizadas, não sendo possível registrar *bugs* com a sessão neste *status*. Uma vez iniciada a sessão, o Usuário passa a ter acesso ao botão Registrar *Bug*, conforme ilustrado na Figura 24.

Figura 24: Tela com sessão em execução (Fonte: Autoria própria).

Durante a execução da sessão, ao detectar um *bug*, o testador deverá clicar no botão Adicionar *Bug*, sendo redirecionado à tela de registro da entidade. Nesta tela estarão disponíveis informações da sessão e inputs para adicionar a descrição do *bug*, criticidade e também um campo para anexar um arquivo, podendo este ser foto ou vídeo, disponível na Figura 25. Isto representa uma evolução significativa da versão anterior, na qual era possível salvar apenas a descrição textual. Essas novas adições fornecem uma representação visual da situação onde o *bug* foi encontrado, permitindo maior entendimento do contexto, repetibilidade e rastreio por parte dos testadores.

Uma vez salvo, o sistema atribui um *id* ao *bug*, bem como a data e hora da detecção, tornando-o único para aquela sessão. Na tela de detalhes (Figura 26), é possível visualizar

Figura 25: Tela de Registro de Bug (Autoria própria).

todos os *bugs* registrados para aquela sessão, gerando uma visão ampla e compreensiva do teste.

Ao finalizar a sessão, não é possível adicionar mais *bugs*, tornando-a imutável, a fim de manter os registros consistentes. A finalização pode ocorrer de maneira manual, ao clicar no botão Finalizar Sessão, ou automaticamente, caso o tempo estipulado seja ultrapassado.

4.1.5 Tela de Listagem de Estratégias

Como apontado anteriormente, esta sessão é de acesso público, não sendo necessário estar registrado no sistema para acessá-la, como visualizado na Figura 27. Nela estarão contidas todas as estratégias cadastradas no sistema em uma única tela, na qual o usuário poderá clicar nos botões para obter mais detalhes sobre a instância, como visto na Figura 27. Caso o usuário seja um administrador, haverá também um botão para adicionar novas estratégias, conforme Figura 28.

A tela de detalhes da estratégia possui uma sessão de informações básicas sobre ela, contendo seu *id*, nome, descrição. Também é possível ver exemplos e dicas sobre o seu uso, a fim de elucidar possíveis dúvidas que o usuário possa vir a ter. Esta documentação é imprescindível para garantir que os testadores saibam se estão aplicando a estratégia da maneira desejada, o que garante uma substancial melhora na qualidade do testes, conforme apresentado na Figura 29.

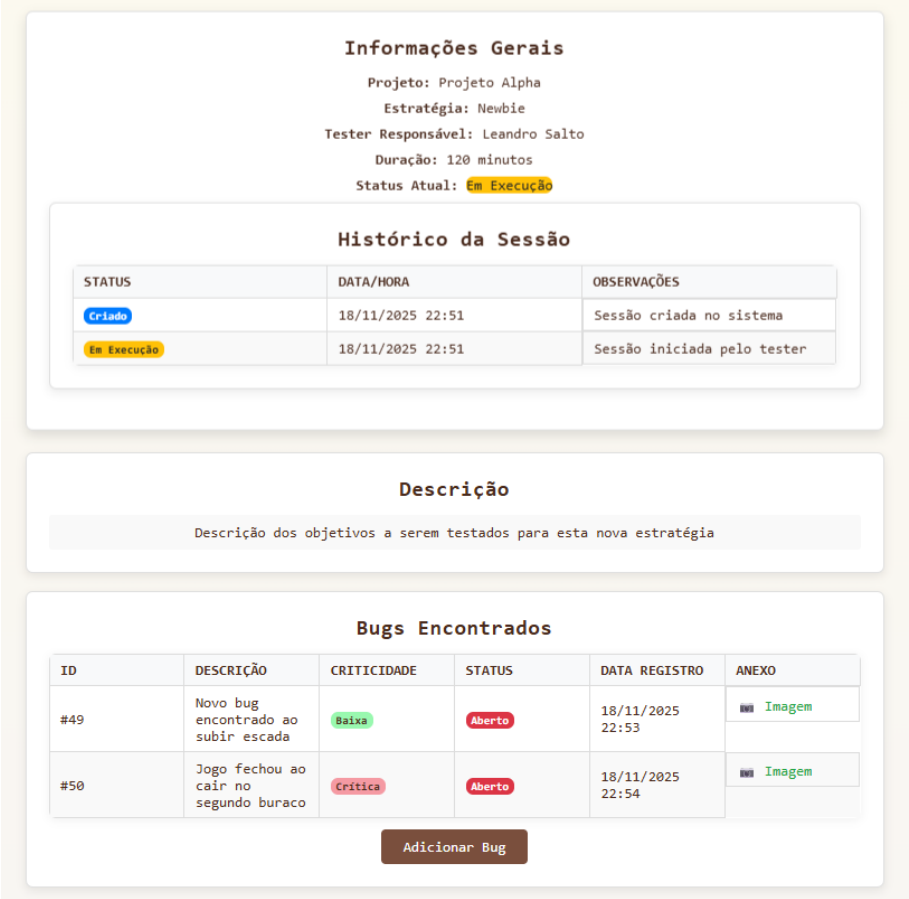

Figura 26: Tela de sessão com *bugs* registrados (Fonte: Autoria própria).

Figura 27: Tela com listagem de estratégias na visão de visitante (Fonte: Autoria própria).

Figura 28: Tela com listagem de estratégias na visão de administrador (Fonte: Autoria própria).

As estratégias podem ter diversos exemplos e dicas registrados, pois a relação entre as classes é de **1:N**, a fim de ter a possibilidade de possuir o máximo de informação possível dentro da mesma tela sobre uma instância da entidade.

Comparando-se com a versão anterior, esta funcionalidade manteve-se relativamente inalterada, pois sua finalidade era a criação, leitura, alteração e deleção de instâncias da entidade Estratégia, porém, houve uma alteração na tela de listagem, para ter um encaixe melhor com a identidade visual do Spring Espresso, além de corrigir o *bug* de visualização das imagens, que não estavam sendo carregadas adequadamente.

4.1.6 Tela de Listagem de *Bugs*

Esta tela representa uma funcionalidade nova no Spring Espresso, a relação global dos *bugs* relatados, ordenados por projeto.

A tela oferece uma forma abrangente do progresso das sessões de teste, possibilitando a identificação de quais detectaram um maior número de *bugs*, ou quais estratégias se mostraram mais eficazes na detecção de problemas.

Nela está contida uma tabela com informações cruciais para uma rápida identificação do *bug* em questão, com o projeto em que ele está alocado, a sua descrição, criticidade, status, data de detecção e a possibilidade de acessar seus detalhes, visível na Figura 30.

Esta composição oferece um potencial ganho na produtividade dos usuários em navegar

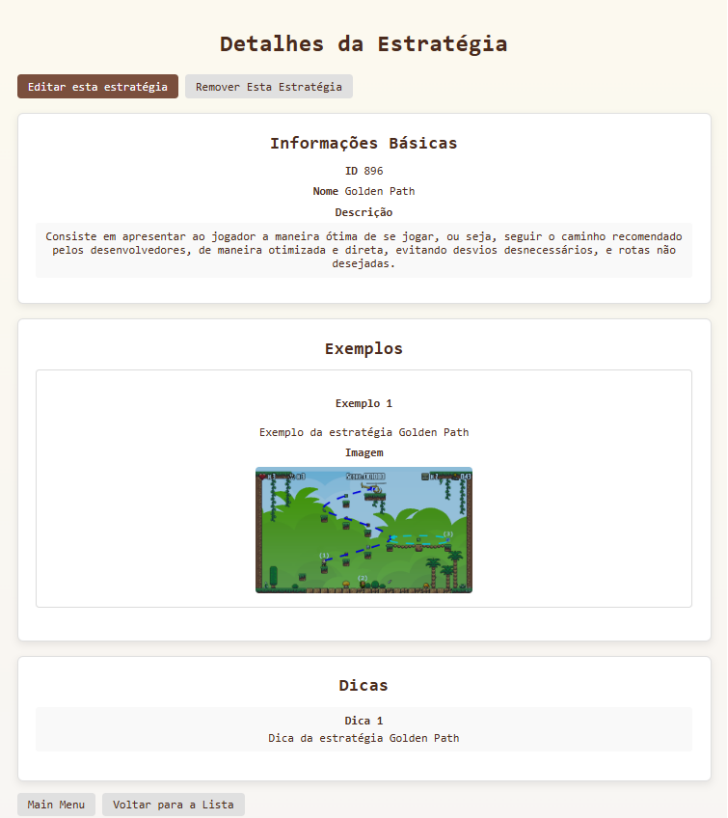

Figura 29: Tela de detalhes das estratégias (Fonte: Autoria própria).

através dos *bugs* do sistema, oferecendo dinamismo às equipes acerca de suas sessões de testes.

Figura 30: Tela com listagem global de *bugs* (Fonte: Autoria própria).

Comparando-se com o protótipo inicial, esta tela representa uma importante evolução, não só para a aplicação do framework *xPloiT*, mas também para qualquer metodologia que seja usada, pois esta forma de visualização global fornece informações desejáveis para todo tipo de abordagem.

4.2 Reflexões Acerca do Desenvolvimento

Na primeira versão, utilizada como base, projetos, sessões, estratégias e *bugs* eram organizados de maneira genérica, com uma maior liberdade de combinações e menos integração entre as entidades. As sessões também eram criadas diretamente dentro de um projeto, porém as estratégias eram entidades independentes, e representavam apenas um atributo da sessão, o que consequentemente distanciava entidade Estratégia como parte integrante da definição de um projeto. Além disso, os *bugs* continham apenas uma descrição textual, o que limitava o seu entendimento e reprodutibilidade, pontos fundamentais para o sucesso de um TE.

Na nova versão, essa estrutura foi reorganizada e é possível notar mais claramente o papel central das estratégias, assim como o *framework xPloiT* propõe. Atualmente:

- Cada **Projeto** contém um conjunto fixo de estratégias baseadas nas estratégias do *framework xPloiT*, definidos no momento de sua criação, não sendo possível alterá-las;
- As **Estratégias** possuem diversas **Sessões** diretamente alinhada a elas, reforçando a noção de que as atividades do testador são guiadas por uma abordagem específica, e que as sessões têm um objetivo claro de como proceder com os testes;
- Uma **Sessão** é um ponto formal de captura de *bugs*, com possibilidade de *upload* de **imagens** ou **vídeos** por parte do testador, adicionando elementos visuais para fácil identificação e reprodução do *bug*. Além disso, também é possível adicionar a criticidade do *bug*, trazendo mais significado a ele de maneira direta.
- Foi criada uma tela de Listagem Global de *Bugs*, que representa um grande avanço para a interpretação dos resultados das sessões de maneira sistêmica, permitindo que as equipes possam retirar importantes reflexões sobre seus testes.

Esta nova abordagem, além de ter fundamentação teórica, baseado em artigos relevantes sobre TEs, como demonstrado na bibliografia, também trouxe mais clareza sobre o contexto das detecções de *bugs*, tornando o resultado final mais confiável e de fácil interpretação pelos usuários.

4.3 Mudanças de Estrutura e Modelagem

Ao longo do processo de desenvolvimento da nova versão do Spring Espresso, a estrutura de classes foi reformulada. As principais mudanças podem ser divididas nos três contextos: reorganização das entidades, melhoria da experiência do usuário na consistência de dados.

4.3.1 Reorganização das classes e papel central da Entidade Estratégia

Um dos ajustes que mais demandaram tempo e esforço foi redesenhar a estrutura de classes e cardinalidades do sistema. Na versão anterior, a entidade **Estratégia** tratava-se de uma entidade praticamente isolada, sendo apenas usada como um atributo de Sessão, não possuindo qualquer vínculo com Projeto, conforme demonstrado na Figura 11. Após a reestruturação, a **Estratégia** passou a ocupar papel central nas relações entre as classes, principalmente por possuir uma relação **N:N** com Projeto, sendo a entidade responsável por organizar e orientar as sessões conforme o conjunto de abordagens definidas pelo *xPloiT*, conforme ilustrado na Figura 31.

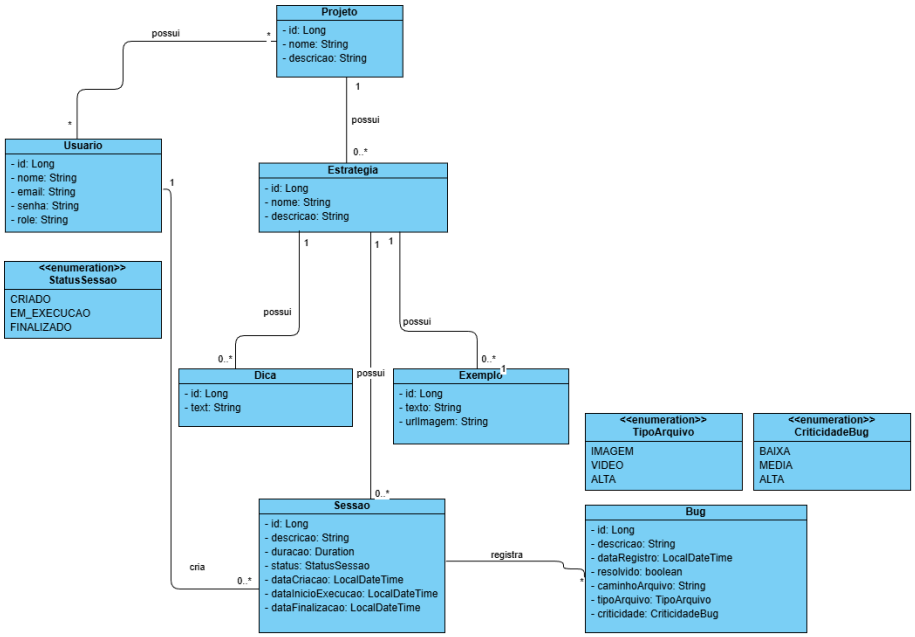

Figura 31: Nova estrutura de classes do Spring Espresso, com a estratégia situada como estrutura central (Fonte: Autoria própria).

No código, esta relação se apresenta da seguinte forma, na classe modelo do Projeto (ProjetoModel.java), ilustrado pela Figura 32:

```
@ManyToMany(fetch = FetchType.LAZY, cascade = {CascadeType.PERSIST, CascadeType.MERGE})
@JoinTable(
        name = "projeto_estrategias",
        joinColumns = @JoinColumn(name = "projeto_id"),
        inverseJoinColumns = @JoinColumn(name = "estrategia_id")
)
private List<EstrategiaModel> estrategias = new ArrayList<>();
```

Figura 32: Código do atributo de lista de estratégias na entidade Projeto (Fonte: Autoria própria).

No modelo da sessão (SessaoModel.java), há a criação dos atributos “projeto” e “estrategia” na classe do modelo da Sessão, conforme visto na Figura 33:

Desta forma, cada projeto agora é instanciado com um conjunto fixo de estratégias, refletindo, mas não se limitando, às nove originais do *xPloiT*, garantindo uniformidade e

```java
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "projeto_id", nullable = false)
private ProjetoModel projeto;

@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "estrategia_id", nullable = false)
private EstrategiaModel estrategia;
```

Figura 33: Código dos atributos “projeto” e “estrategia” na entidade Sessao (Fonte: Autoria própria).

alinhamento na forma como diferentes projetos utilizam o sistema. Isto facilitou comparar sessões originadas da mesma estratégia e entender melhor o contexto em que cada *bug* foi encontrado.

Por meio disto, podemos garantir que uma estratégia possa ser reutilizada em diversas sessões, porém a sessão se limitará a manter uma única instância da estratégia, mais uma vez reforçando a importância desta entidade.

Outra mudança foi a alteração de uma relação previamente existente entre Sessão e Historico Status, que seria a relação de histórico de edição de uma sessão. No modelo anterior, apesar de válida, esta abordagem resultava em uma estrutura excessivamente fragmentada, em um domínio no qual não era exigida, portanto, na versão atual, a entidade Sessão possui dois novos atributos indicando a data e hora de execução e a data e hora de finalização, conforme ilustrado pela Figura 34.

```java
@Column(name = "data_criacao", nullable = false)
private LocalDateTime dataCriacao;

@Column(name = "data_inicio_execucao")
private LocalDateTime dataInicioExecucao;

@Column(name = "data_finalizacao")
private LocalDateTime dataFinalizacao;
```

Figura 34: Atributos referente às datas de criação, início de execução e finalização na classe modelo de Sessão (Fonte: Autoria própria).

Esta alteração resultou em simplicidade arquitetural, o que trouxe benefícios ao reduzir o acoplamento entre as entidades e tornar mais clara a leitura e manutenção do código, sem perder a funcionalidade de rastreio de status. Esta simplificação também tornou possível algo que era desejado que ocorresse já na versão de protótipo, que era a finalização automática de sessões.

Esta funcionalidade passou a existir na nova versão, pois o criador da sessão estipula um tempo limite para a execução dos testes, e uma vez iniciada, o sistema iria verificar se o tempo entre o início da execução e o horário atual já foi ultrapassado a cada 20 segundos e, em caso positivo, irá alterar o status da sessão automaticamente para "finalizado".

Na tela, o sistema identifica se a sessão foi finalizada automaticamente ou não, aumentando consideravelmente a consistência de informação, ao passo que na versão anterior era possível passar o limite de tempo indefinidamente, além de ser possível registrar *bugs* independente do status da sessão, que também foi corrigido na nova versão.

A fim de se facilitar a refatoração do código. Houve a substituição do tipo *UUID* para os *ids* das entidades para o tipo *Long*, pois no decorrer das alterações, o primeiro tipo apresentou algumas incompatibilidades com o Spring Data JPA e MySQL, o que poderia ter sido contornado, mas devido à necessidade de simplificação nesta etapa, optou-se por substituí-lo. Apesar do antigo tipo ser mais robusto, o tipo *Long* é mais intuitivo e simples de se analisar no desenvolvimento, o que acelerou grandemente o andamento do *redesign* do sistema.

Um benefício indireto da reorganização da estrutura de classes foi o aumento de consistência de dados, pois como agora as estratégias são uma entidade centralizadora, não é mais possível criar sessões que não possuam estratégias associadas, o que vai totalmente a favor da implementação da metodologia *xPloiT* nos TEs.

4.4 Ajustes de comportamentos, layout e elementos da página

Diversas melhorias visuais e estruturais foram implementadas no desenvolvimento da nova versão a fim de tornar a interface do usuário mais intuitiva, eficiente e agradável. Dentre elas, destacam-se o aumento da largura das colunas das tabelas, bem como limitar o texto a ficar dentro dela, permitindo uma leitura mais confortável, reorganização dos botões, a fim de se reduzir os cliques necessários para realizar tarefas.

Também se agrupou funcionalidades semelhantes, tornando a navegação fluida, e atributos não relevantes ao usuário foram removidos das tabelas, tais quais identificadores internos.

4.4.1 Prevenção de fluxo incorreto de ações

Na versão anterior do sistema, diversos comportamentos indesejados foram identificados, os quais comprometiam o fluxo correto de execução da ferramenta. Entre eles, destacavam-se a possibilidade de registrar *bugs* em sessões que não estavam em execução, permitindo que um administrador conseguisse se excluir do sistema, possibilitando de que não houvesse nenhum administrador.

Além disso, os visitantes não tinham acesso aos detalhes das estratégias, o que restringia a obtenção de informação de pessoas externas ao sistema, de maneira indesejada.

Também foi corrigido o problema de exibição das imagens, as quais ficavam apenas representadas pelo seu link, mas não era possível ver o seu conteúdo, o que impedia a representação visual de exemplos e dicas das estratégias.

4.5 Lições Aprendidas

O processo de reconstrução do sistema se revelou significativamente mais árduo do que o inicialmente previsto. Diversas dificuldades foram encontradas durante a refatoração, e estas reforçaram uma lição da Engenharia de Software: **uma modelagem mal definida, mesmo que apenas em um ponto específico do sistema, tem potencial para desencadear uma reação em cadeia que compromete toda a aplicação**. Na realidade, ao tentar alterar as classes uma a uma, quando já estavam todas relacionadas e funcionando, mostrou-se uma má abordagem, pois a cada alteração feita, mesmo que aparentemente pequenas, desencadeavam falhas graves no sistema, na maioria das vezes, ocasionando na parada total do software.

Pensando nisso, em determinado momento da implementação, resolveu-se mudar a abordagem: **Reiniciar o desenvolvimento**, desta vez, partindo-se de uma modelagem mais sólida e cuidadosamente planejada.

A reformulação recomeçou pelas entidades Estratégia e Projeto. A partir delas, todas as demais dependências foram adicionadas de forma incremental, sendo relacionadas de maneira correta e de forma que fosse se integrar com as que estavam por vir. Esta abordagem modular, juntamente com o pré-planejamento das demais estruturas, mostrou-se decisiva para o sucesso da refatoração, pois somente após esta decisão houve avanço significativo e aumento de velocidade de desenvolvimento, gerando uma base sólida para se construir, e eliminar inconsistências geradas ao se alterar as classes que já estavam estruturadas de outra forma.

Após a finalização do projeto, ficou evidente que esta estratégia deveria ter sido adotada anteriormente, pois muito tempo foi desperdiçado em ações que chegavam ao ponto de nada mais funcionar no sistema, tendo de ser desfeitas ou totalmente reformuladas. Caso tivesse adotado a estratégia de começar a estruturar as classes uma a uma, grande parte do retrabalho e dos empecilhos teria sido evitada. É possível afirmar que, com uma base robusta desde o início, o desenvolvimento teria evoluído muito mais rapidamente, e com mais qualidade, o que certamente teria permitido a inclusão de funcionalidades adicionais desejáveis ao escopo deste trabalho.

Ainda assim, apesar de desgastante, o processo trouxe valiosos aprendizados sobre boas práticas de engenharia de software, planejamento, e importância de simplicidade estrutural, reforçando que sistemas mais complexos exigem conceitos sólidos e bem estruturados, e que compensa investir tempo na definição correta das relações das classes e entidades, para que a implementação possa correr de forma fluida e produtiva.

5 Conclusão

O *redesign* do Spring Espresso permitiu que limitações arquiteturais e conceituais da versão original fossem superadas, fornecendo uma base técnica sólida e mais alinhada ao *framework xPloiT*. As modificações resultaram num modelo mais claro, eficiente, escalável.

A decisão de reiniciar o desenvolvimento mostrou-se essencial para evitar conflitos e garantir um desenvolvimento estruturado ao sistema. Como resultado, o sistema passou a oferecer suporte adequado ao seu objetivo principal, ao mesmo tempo que possui uma estrutura escalável para futuras melhorias, tais como análises de métricas e funcionalidades colaborativas. Desta forma, o trabalho demonstra que não apenas foram solucionados problemas pré-existentes, mas também houve ganho de funcionalidades importantes para garantir que o Spring Espresso evolua de forma relevante.

5.1 Limitações

Apesar do Spring Espresso possuir uma boa estrutura de apoio à realização de TEs, ela ainda carece de ferramentas de análise, o que limita a avaliação profunda dos resultados das sessões de teste. A ausência de gráficos, dashboards e relatórios nativos deixam a desejar para a obtenção de métricas essenciais, tais como quais estratégias captam mais *bugs*, qual testador tem um melhor aproveitamento de tempo de sessão ou qual o tempo médio para a detecção de falhas.

Também há limitação no tamanho de arquivos possíveis para fotos e vídeos, que podem ser otimizados para que seja possível salvar fotos em alta definição e vídeos com mais de 10MB, sem que haja perda de qualidade ou erros ao salvar.

Além disso, a ferramenta tem forte dependência da entrada manual de informação por parte do usuário, o que pode acarretar em erros humanos, e inconsistência de informações, apesar dos esforços tomados para minimizá-los.

Outra limitação detectada está na impossibilidade de se alterar as estratégias alocadas a um projeto, reduzindo a liberdade dos administradores de gerenciar um projeto inicializado.

Adicionalmente, não existem formas integradas de se interagir com o sistema, havendo a necessidade do testador parar com o fluxo de testes para registrar *bugs*, trazendo falta de eficiência no processo.

5.2 Trabalhos Futuros

Os trabalhos futuros representam aprimoramentos incrementais e etapas essenciais para consolidar o Spring Espresso como uma solução para equipes que utilizam TE no desenvolvimento de jogos. As limitações supracitadas podem ser supridas através de melhorias adicionadas ao sistema, tais como:

- Possibilidade de se atualizar quais estratégias estão disponíveis para um projeto depois de criado;
- Proporcionar salvar fotos em alta definição, ou vídeos com mais de 10MB;
- Diminuir número de cliques para realizar ações repetitivas;

Além disso, novas funcionalidades podem ser implementadas, que poderiam beneficiar a ferramenta, por exemplo:

- Possibilidade de criação de dashboards analíticos com métricas de sessões e estratégias;
- Exportação automática de relatórios em PDF;
- Aprimoramento da gamificação e do sistema de medalhas, como premiações por *bugs* encontrados e resolvidos;
- Suporte para sessões colaborativas;
- Adição de elementos de integração direta com o navegador, como tecla de atalhos para salvar imagens e vídeos durante a execução das sessões de teste.

5.3 Considerações finais

Este estudo teve como objetivo reestruturar e aprimorar o Spring Espresso, ferramenta voltada ao apoio a TEs, corrigindo problemas conceituais e estruturais da versão original e criando uma base sólida para a evolução futura, focada na aplicação do *framework xPloiT*.

A refatoração buscou resolver problemas e suprir lacunas do modelo antigo, principalmente posicionando a entidade **Estratégia** como elemento central na estrutura da aplicação, corroborando com a estratégia adotada pelo *framework*. Esta mudança perimitiu uma modelagem feita sob medida para o objetivo proposto, além de favorecer clareza entre projeto, sessões e demais artefatos. Além disso, a entidade Historico Status foi substituídas por soluções mais simples e tão eficazes quanto, reduzindo fragilidades e aumentando desempenho e legibilidade do código.

No processo de reconstrução, constatou-se que reparos incrementais visando uma reformulação na estrutura de classes num sistema já funcional produziam novos conflitos e inconsistências que dificultavam o avanço contínuo do desenvolvimento. A decisão de reiniciar o desenvolvimento a partir de um ponto inicial bem definido foi de suma importância para o sucesso do projeto, uma vez que possibilitou a construção de uma base sólida, na qual todas as outras entidades foram acopladas e integradas, funcionando de maneira desejável.

Com base nisso, o trabalho apresentado entrega uma ferramenta com forte fundamentação teórica no contexto de TEs, com uma base técnica, possuindo um modelo conceitual claro e escalável, capaz de sustentar futuras melhorias, por exemplo, a adição de funcionalidades de obtenção de métricas importantes para análises de eficiência, ou suporte a sessões colaborativas.

Em síntese, o projeto contribui para o entendimento de desafios práticos no desenvolvimento de software, além de oferecer uma nova arquitetura que viabiliza a expansão da ferramenta. Os resultados indicam que a refatoração realizada não apenas solucionou limitações conhecidas, mas também criou uma base sólida para que o Spring Espresso evolua como ferramenta de TEs de forma relevante e válida para o desenvolvimento de softwares.

Referências

1 BACH, J. *Exploratory testing explained*. [S.l.: s.n.], 2003.

2 ITKONEN, J. Empirical studies on exploratory software testing. Aalto University, 2011.

3 DUARTE, Y.; POLITOWSKI, C.; ENDO, A. T. Towards a framework for exploratory testing in video games. In: IEEE. *2025 IEEE/ACM 9th International Workshop on Games and Software Engineering (GAS)*. [S.l.], 2025. p. 25–32.

4 LYNDSAY, J.; EEDEN, N. V. Adventures in session-based testing. *Workroom Productions Ltd*, v. 27, n. 5, p. 1–17, 2003.

5 WHITTAKER, J. A. *Exploratory software testing: tips, tricks, tours, and techniques to guide test design*. [S.l.]: Pearson Education, 2009.

6 CASTRO, A. K. S. d. Testes exploratórios: Características, problemas e soluções. Universidade Federal do Rio Grande do Norte, 2018.

7 DUARTE, Y. et al. Exploratory testing for platform video games: strategies and lessons learned. In: . [S.l.: s.n.], 2024. v. 15, n. 1, p. 657–669.

8 WIDDER, D. G.; GOUES, C. L. What is a"bug"? on subjectivity, epistemic power, and implications for software research. *arXiv preprint arXiv:2402.08165*, 2024.

9 SILVA, F. F. da et al. User experience techniques adopted by brazilian game development studios: An exploratory study. In: . [S.l.]: Wiley Online Library, 2024. v. 2024, n. 1, p. 1857881.

10 MICALLEF, M.; PORTER, C.; BORG, A. Do exploratory testers need formal training? an investigation using hci techniques. In: IEEE. *2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. [S.l.], 2016. p. 305–314.

11 COPPOLA, R. et al. On effectiveness and efficiency of gamified exploratory gui testing. In: . [S.l.]: IEEE, 2023. v. 50, n. 2, p. 322–337.

12 COUTINHO, J. C.; ANDRADE, W. L.; MACHADO, P. Implementing exploratory testing in an agile context: A study based on design science research. p. 67–76, 2023.

13 LEVEAU, J. et al. Fostering the diversity of exploratory testing in web applications. *Software Testing, Verification and Reliability*, Wiley Online Library, v. 32, n. 5, p. e1827, 2022.

14 CAVNAR, W. B.; TRENKLE, J. M. et al. N-gram-based text categorization. In: LAS VEGAS, NV. *Proceedings of SDAIR-94, 3rd annual symposium on document analysis and information retrieval*. [S.l.], 1994. v. 161175, p. 14.

15 VMware, Inc. *Spring Boot Documentation*. 2025. Acesso em: 11 nov. 2025. Disponível em: <https://spring.io/projects/spring-boot>.

16 TEAM, T. *Thymeleaf Documentation*. 2025. Acesso em: 11 nov. 2025. Disponível em: <https://www.thymeleaf.org/documentation.html>.

17 CORPORATION, O. *MySQL 8.0 Reference Manual*. 2025. Acesso em: 11 nov. 2025. Disponível em: <https://dev.mysql.com/doc/>.

18 VMware, Inc. *Spring Data JPA Documentation*. 2025. Acesso em: 11 nov. 2025. Disponível em: <https://spring.io/projects/spring-data-jpa>.