SERVIÇO PÚBLICO FEDERAL · MINISTÉRIO DA EDUCAÇÃO

UNIVERSIDADE FEDERAL DE VIÇOSA · UFV

CAMPUS FLORESTAL

CCF 323 - Arquitetura de Software

Testes Funcionais, Estruturais e de Mutação- Sprint 03

Equipe 5º Ano - “BYTECRAFT”

Matheus e Edgar

Florestal - MG

2025

SUMÁRIO

Testes estruturais (caixa branca)

Bug em `AlunoDTO.java`

Não se sabe se por testes funcionais daria para replicar esse problema aqui, mas através dos testes estruturais foi identificado esse problema. Basicamente, o problema se daria se tivesse um aluno que concluísse todo o jogo e, na hora que ele visse o ranking, se tivesse um aluno cadastrado que ainda não escolheu o nível, poderia quebrar o sistema de ranking por conta desse aluno que ainda não escolheu o sistema de nível.

O cenário do bug é assim:

1. Aluno A se cadastra no sistema, vincula à sala, mas ainda não escolheu a dificuldade (fácil/médio/difícil)
2. Aluno B joga normalmente, escolhe dificuldade, completa o Modo História ou Modo Quiz
3. Aluno B clica no botão "Ver Ranking da Turma" após finalizar o jogo
4. O backend busca TODOS os alunos daquela sala para montar o ranking
5. Quando tenta converter o Aluno A para AlunoDTO, o código tenta acessar aluno.getNivel().name()
6. BUG- NullPointerException porque o Aluno A não tem nível definido ainda (nivel é null)
7. O sistema crasha, o Aluno B não consegue ver o ranking e recebe um erro 500
8. O bug só acontece se existir pelo menos um aluno na mesma sala que ainda não escolheu a dificuldade. Se todos os alunos da sala já tiverem escolhido o nível, o ranking funciona normalmente.

É um problema porque é comum ter alunos em diferentes estágios - uns já jogando, outros apenas cadastrados. A correção que fizemos aluno.getNivel() != null ? aluno.getNivel().name() : null, permite que o sistema mostre o ranking mesmo com alunos "incompletos", simplesmente deixando o campo nível como null para eles.

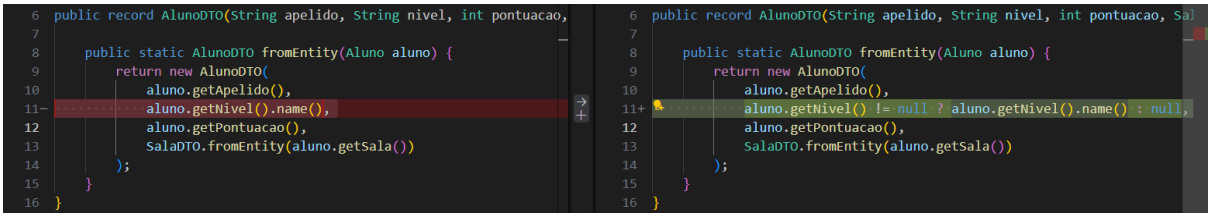


Figura 1: Solução proposta

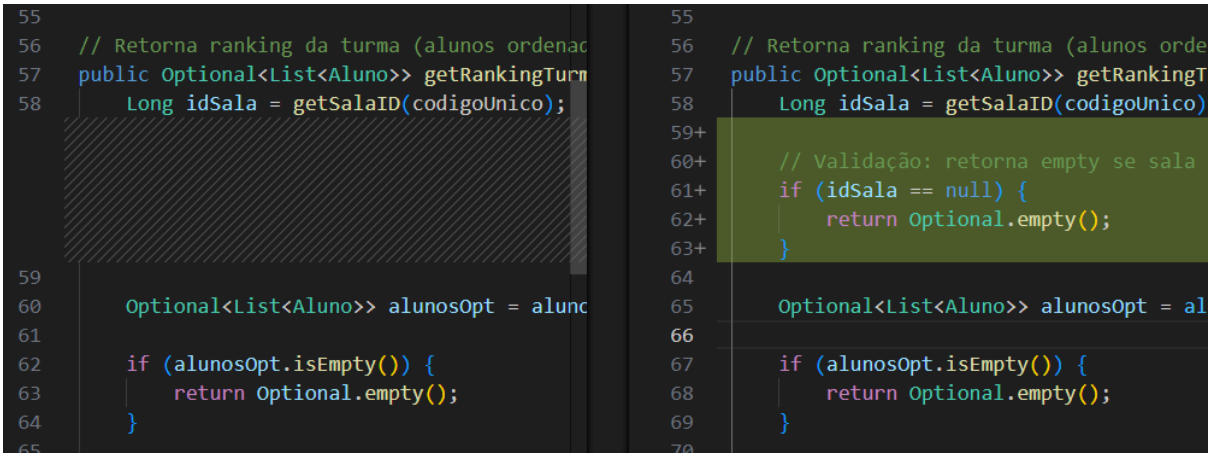
Bug em `SalaService.java`

O segundo problema se dá em buscar o ranking de uma sala que não existe. Se por algum motivo o sistema buscar o ranking de uma sala que não existe, ele pode travar. Então, colocar uma validação seria importante.

Então o problema é que o sistema pegava esse código, não encontrava nada no banco de dados e recebia um `null` de volta. O problema é que o código continuava tentando trabalhar com esse `null` como se fosse um ID válido, passando ele adiante para outras partes do sistema. Isso cria um comportamento imprevisível e pode causar erros estranhos que são difíceis de debugar depois.

A solução foi adicionar uma validação logo no início do método getRankingTurma. Colocamos um if (idSala == null) { return Optional.empty(); } que verifica se o ID da sala existe antes de tentar fazer qualquer coisa com ele. Assim, quando uma sala não existe, o sistema retorna um resultado vazio, sem tentar processar dados inválidos. Para validar essa correção, rodamos .\mvnw.cmd test -Dtest="SalaServiceStructuralTest" e os 17 testes passaram com sucesso. O teste específico que valida sala inexistente agora confirma que o método retorna vazio corretamente, sem chamar o repositório com valores nulos.

No final, rodamos toda a suite de testes com (lembre se de estar na pasta do backend, ou passar o caminho da backend) .\mvnw.cmd test -Dtest="AlunoServiceStructuralTest,SalaServiceStructuralTest,AlunoDTOStructuralTest,AlunoServiceMutationTest", e os 57 testes passaram. Também refizemos os testes com essas correções propostas, e todos passaram, mostrando que as correções são efetivas, porém seria interessante fazer mais alguns testes funcionais depois dessas alterações do código. Por mais que sejam mínimas, isso ajuda a garantir a qualidade do produto.



```
55
56  // Retorna ranking da turma (alunos ordenac
57  public Optional<List<Aluno>> getRankingTurm
58      Long idSala = getSalaID(codigoUnico);
59
60      Optional<List<Aluno>> alunosOpt = alunc
61
62      if (alunosOpt.isEmpty()) {
63          return Optional.empty();
64      }
```

```
55
56  // Retorna ranking da turma (alunos orde
57  public Optional<List<Aluno>> getRankingT
58      Long idSala = getSalaID(codigoUnico)
59+
60+     // Validação: retorna empty se sala
61+     if (idSala == null) {
62+         return Optional.empty();
63+     }
64
65      Optional<List<Aluno>> alunosOpt = al
66
67      if (alunosOpt.isEmpty()) {
68          return Optional.empty();
69      }
```

Figura 2: Solução proposta

Testes funcionais (caixa preta)

O presente relatório tem como objetivo validar o funcionamento do processo em que o aluno, dentro do Modo História, posiciona corretamente as peças na montagem de um computador. O teste contemplou o fluxo principal, o fluxo alternativo e as regras de negócio associadas. Durante a execução, foram avaliadas diferentes situações, níveis de dificuldade (Fácil, Médio e Difícil) e condições de sucesso e falha, verificando a exibição de mensagens, cálculos de pontuação e registro de progresso.

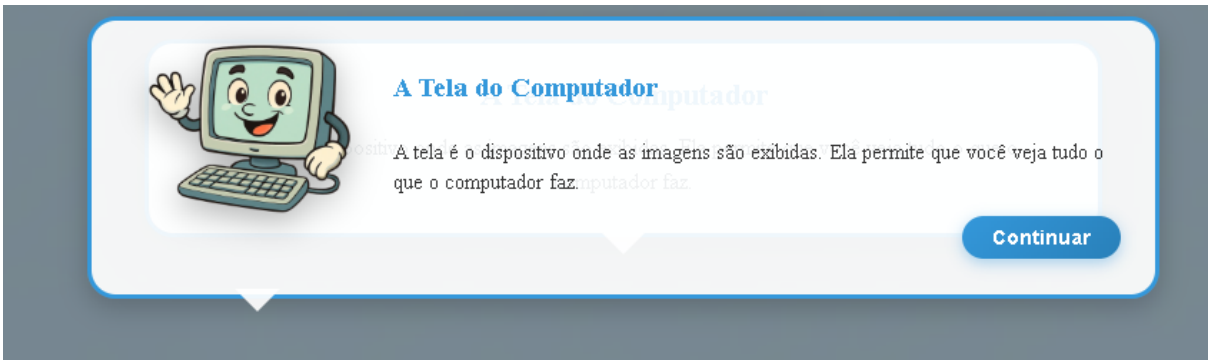
Inicialmente, foi acessado o Modo História conforme descrito no fluxo principal. O sistema apresentou corretamente a história introdutória e a lista de peças disponíveis. A seleção das peças e o carregamento da área de encaixe ocorreram de forma adequada, permitindo ao aluno interagir com o ambiente de montagem.

Durante os testes de posicionamento correto da peça, o sistema exibiu a mensagem de sucesso, e o progresso do aluno foi devidamente registrado. No entanto, foram observados alguns comportamentos divergentes e problemas visuais que impactam parcialmente a experiência do usuário.

Resultados Observados

1. Comportamento de Interface e Modais

Durante a execução, ao selecionar o botão “Continuar”, dois modais foram acionados simultaneamente. Quando o usuário opta por continuar, ambos os modais desaparecem ao mesmo tempo.



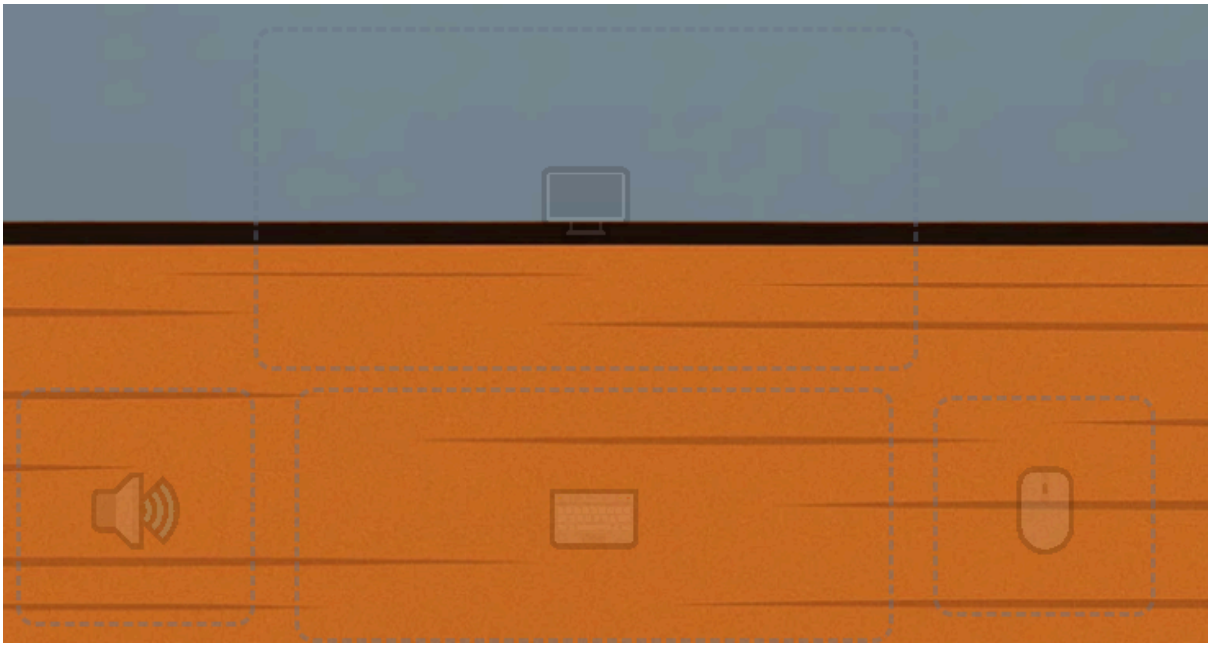
2. Sprites e Elementos Gráficos

Foi identificado que os sprites dos teclados estão diferentes entre si, apresentando inconsistências visuais no design.

Além disso, o teclado azul posicionado aparece em tamanho reduzido, destoando do restante dos elementos da montagem.

3. Exibição do Símbolo da Dropzone

Durante os testes de todas as dificuldades (Fácil, Médio e Difícil), observou-se que o símbolo da dropzone com o ícone representativo da peça continua visível ao jogador independente do nível selecionado.

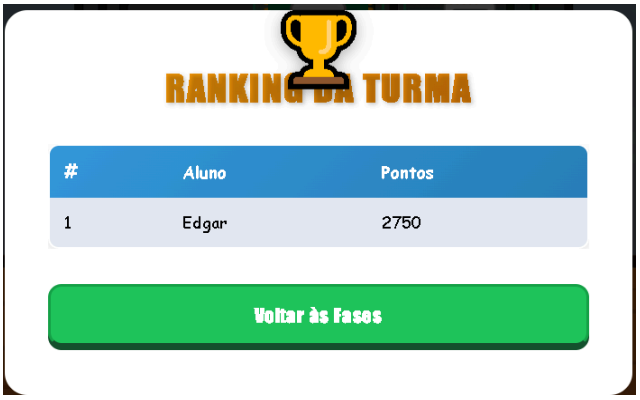


4. Nível de Montagem Interno

No nível de montagem interno, nenhum ícone ou marcação de encaixe é exibido, tornando a interação menos intuitiva. Para alunos que não possuem conhecimento prévio sobre o posicionamento interno dos componentes de um computador, a dedução de onde cada peça deve ser encaixada se torna bastante difícil.

5. Cálculo e Exibição da Pontuação

Foram detectadas inconsistências no cálculo e na exibição da pontuação:
A pontuação final exibida ao aluno não corresponde à pontuação mostrada no ranking da turma.A soma total das pontuações exibidas durante os encaixes das peças não é compatível com o valor final apresentado.
Esses problemas sugerem falhas na aplicação das regras de negócio RN10 (cálculo por tentativas) e RN22 (fórmula de bonificação por tempo). É necessário revisar a lógica de cálculo e a sincronização dos valores armazenados no banco de dados e apresentados nas interfaces.



6. Retorno à Tela de Fases

Ao testar o botão “Voltar às fases”, verificou-se que o sistema retorna corretamente à tela de fases, porém o nível exibido volta automaticamente para o modo Fácil, independentemente do nível selecionado anteriormente. Além disso, essa alteração de nível não é registrada no banco de dados, sendo possível jogar o nível fácil armazenando a pontuação como difícil no banco de dados.