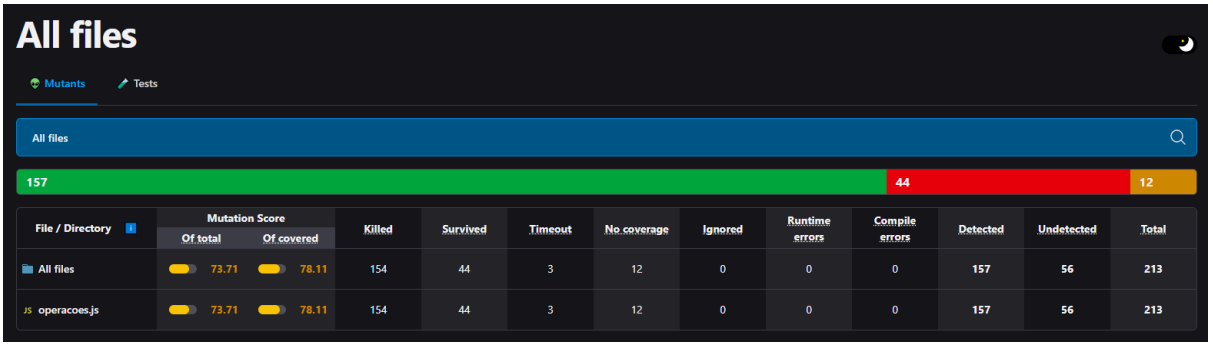
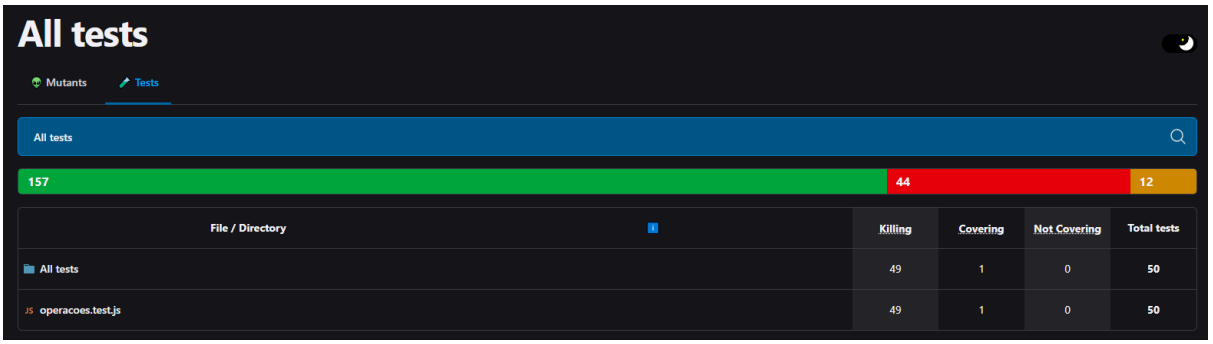
Testes de Software

Análise de Eficácia de Testes com Teste de Mutação

Lucas Ferreira Garcia
804803

1. **Análise Inicial**





Cobertura de código (Jest): 100%.
Mutation Score (Stryker, execução inicial): 73,71%.

Resumo dos resultados:

- 213 mutantes gerados.
- 154 mutantes mortos.
- 44 mutantes sobreviventes.
- 3 mutantes com timeout.
- 12 sem cobertura.

Análise: Apesar da cobertura elevada (100%), o mutation score foi de apenas 73,71%, indicando que os testes executam o código, mas não verificam corretamente o comportamento em diversos casos. Isso mostra a diferença entre cobertura de execução e eficácia de verificação.

2. **Análise de mutantes críticos:**

A seguir são analisados três mutantes sobreviventes da primeira execução do Stryker:

Mutação 1 - Divisão por 0 (linha 8)

```
function divisao(a, b) {
  if (b === 0) throw new Error('Divisão por zero não é permitida.');
  return a / b;
}
```

Mutação gerada:

- if (b === 0) throw new Error('Divisão por zero não é permitida.');
+ if (b === 0) throw new Error("");

Descrição:

O mutante substituiu a mensagem de erro lançada por uma string vazia. Como o teste original apenas verificava se uma exceção era lançada, e não validava a mensagem, o mutante sobreviveu.

Motivo da sobrevivência:

O teste apenas verifica se uma exceção é lançada, sem comparar o conteúdo da mensagem de erro. Assim, o mutante que altera o texto da exceção passa despercebido

Mutante 2 - Raiz quadrada de número negativo (linha 13)

```
function raizQuadrada(n) {
  if (n < 0) throw new Error('Não é possível calcular a raiz quadrada de um número negativo.');
  return Math.sqrt(n);
}
```

Mutação gerada:

- if (n < 0) throw new Error('Não é possível calcular a raiz quadrada de um número negativo.');
+ if (false) throw new Error('Não é possível calcular a raiz quadrada de um número negativo.');

Descrição:

O Stryker substituiu a condição n < 0 por false, eliminando a verificação para números negativos. Assim, raizQuadrada(-9) retornaria NaN sem lançar erro.

Motivo da sobrevivência:

O teste existente apenas verificava casos válidos, não há verificação para casos negativos, portanto a alteração não foi detectada

Mutante 3 - Função IsPrimo (Linha 73-75)

```
function isPrimo(n) {
  if (n <= 1) return false;
  for (let i = 2; i < n; i++) {
    if (n % i === 0) return false;
```

Mutação gerada:

- if (n <= 1) return false;
+ if (n <= 1) return true;

Descrição:

O mutante inverteu a lógica da verificação inicial, fazendo a função retornar true

para números menores ou iguais a 1, que não são primos.
Motivo da sobrevivência:
O teste original apenas verificava números primos conhecidos, não incluindo valores de borda ou compostos.

3. **Soluções implementadas:**

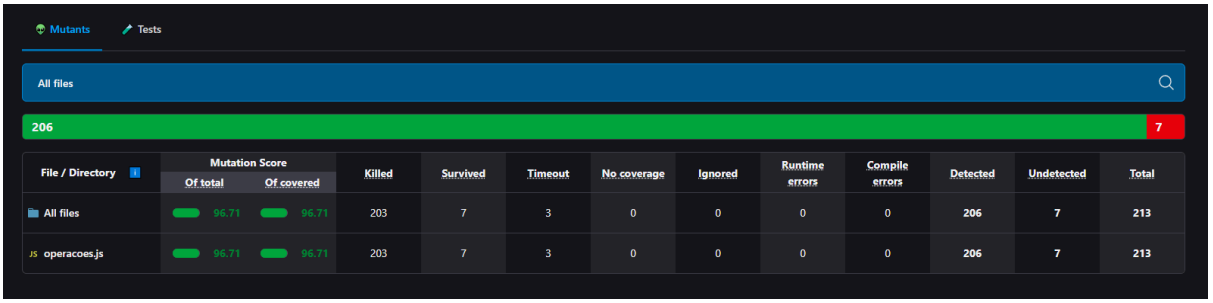
Para atingir melhorar o número de mutantes mortos, foram adicionados casos de teste específicos visando os mutantes que haviam sobrevivido. Os principais novos testes implementados incluem:

1. **Fatorial**: casos para n = 0, n = 1 e n > 1, garantindo cobertura completa de operadores lógicos e condicionais.
2. **Média e Produto de Arrays**: testes com arrays vazios, de um elemento e com valores negativos ou zero, cobrindo todos os ramos condicionais.
3. **Clamp**: testes para valores iguais aos limites e fora do intervalo, garantindo cobertura dos operadores <= e >=.
4. **Paridade e Comparações**: testes com números pares, ímpares e casos de igualdade/diferença, eliminando mutantes de comparação.
5. **Mediana, Máximo e Mínimo**: testes com arrays vazios, pares, ímpares e negativos, cobrindo todos os caminhos condicionais.
6. **Conversão de Temperatura e Inverso**: testes com valores extremos e inválidos, garantindo resultados corretos e exceções apropriadas.

Esses novos testes são eficazes porque garantem que todas as condições lógicas, expressões condicionais e operadores críticos das funções sejam exercitados, eliminando os mutantes sobreviventes e aumentando a confiabilidade da suíte de testes. Para ver a implementação dos novos testes está no arquivo operacoes.test.js

4. **Resultados finais:**

Mutantes Equivalentes:
Os 7 mutantes que permaneceram são mutantes equivalentes, ou seja, suas alterações não afetam o comportamento observável do código para entradas válidas.
- **Fatorial (n === 0 || n === 1):** mudanças na expressão condicional não alteram o resultado para 0, 1 ou valores maiores, que já estão cobertos pelos testes.
- **Produto de array (numeros.length === 0):** mesmo alterando a condição, o retorno correto para arrays vazios, de um elemento ou com zero se mantém.
- **Clamp (< para <= e > para >=):** todos os casos críticos (valores abaixo do mínimo, acima do máximo e nos limites) já estão testados, mantendo o comportamento correto.
Esses mutantes não podem ser mortos sem alterar a lógica da função.



Resultados:

Após o fortalecimento da suíte de testes, os resultados melhoraram significativamente em comparação à execução inicial. O Mutation Score no Stryker subiu de 73,71% para 96,71%, passando de 154 mutantes mortos para 203. O número de mutantes sobreviventes caiu de 44 para apenas 7, todos mutantes equivalentes que não podem ser mortos sem alterar a lógica da função. O número de mutantes com timeout se manteve em 3 e os mutantes sem cobertura reduziram de 12 para 0. Esses resultados demonstram que os novos testes foram eficazes em eliminar a grande maioria dos mutantes, garantindo maior robustez da suíte de testes.

5. Conclusão:

O teste de mutação se mostrou uma ferramenta poderosa para avaliar a eficácia da suíte de testes. Ao introduzir mutantes é possível identificar lacunas na cobertura e nos casos testados, evidenciando situações que poderiam gerar erros não detectados. No nosso caso, os testes adicionais conseguiram “matar” a maioria dos mutantes, aumentando significativamente o mutation score e demonstrando que a suíte agora valida corretamente as regras do sistema. Além disso, a identificação de mutantes equivalentes reforça a compreensão de quais partes do código possuem comportamento intrinsecamente seguro ou imutável frente às alterações, fornecendo uma visão mais precisa da qualidade dos testes. Em resumo, o teste de mutação complementa a cobertura tradicional, assegurando maior confiabilidade e robustez do software.