1. Cenário geral do módulo

O módulo que será testado é um sistema desenvolvido em JavaScript para calcular as notas acadêmicas de alunos em disciplinas semestrais. Este sistema tem como objetivo auxiliar professores e instituições de ensino a determinar o status acadêmico dos alunos com base em suas notas ao longo do semestre.

O que o módulo é e o que faz:

- **Entrada de Dados:** Permite que os usuários (professores ou administradores) insiram as duas notas obtidas pelo aluno durante o semestre. As notas devem ser valores numéricos entre **0** e **10**, inclusos.
- **Validação de Notas:** O sistema verifica se as notas inseridas estão dentro do intervalo válido e se são números reais.
- **Cálculo do Total de Pontos:** Soma as duas notas para calcular o total de pontos do aluno no semestre.
- **Determinação do Status do Aluno:**
  - **Aprovação Direta:** Se o aluno obtiver **12 pontos ou mais** no total e **nenhuma** das notas for menor que **6**, o aluno é aprovado sem necessidade de provas adicionais.
  - **Prova Extra:** Se o aluno tiver **12 pontos ou mais**, mas **alguma das notas** for **menor que 6**, o aluno terá a oportunidade de fazer uma prova extra para melhorar sua nota.
  - **Reprovação:** Se o aluno não atingir **pelo menos 12 pontos** no total, é reprovado na disciplina.
- **Saída de Dados:** O sistema apresenta ao usuário o status final do aluno (aprovado, reprovado ou necessidade de prova extra) com base nas regras estabelecidas.

Principais Funcionalidades a serem Testadas:

- **Funcionalidade de Entrada de Notas:**
  - Verificação da aceitação de valores dentro do intervalo permitido (0 a 10).
  - Rejeição e tratamento adequado de valores fora do intervalo ou de tipos de dados inválidos (e.g., letras ou símbolos).
- **Funcionalidade de Validação de Dados:**
  - Garantia de que valores limites (0 e 10) são aceitos corretamente.
- **Funcionalidade de Cálculo:**
  - Cálculo correto do total de pontos a partir das duas notas inseridas.
  - Precisão em operações aritméticas com números decimais.
- **Funcionalidade de Determinação do Status:**

- Aplicação correta das regras para aprovação, reprovação ou necessidade de prova extra.
- Teste de todos os cenários possíveis, incluindo casos limites (e.g., notas exatamente iguais a 6 ou total de pontos exatamente 12).

2. Estratégia(s) de Teste (como será testado)

2.1. Teste Funcional (Caixa Preta)

Objetivo: Verificar se o sistema atende aos requisitos funcionais especificados, testando as funcionalidades sem considerar a estrutura interna do código.

Técnicas Aplicadas:

- **Particionamento de Equivalência:**
  - **Critério:** Dividir os dados de entrada em classes de equivalência (válidas e inválidas) onde o sistema deve tratar todos os valores de maneira semelhante.
  - **Finalidade:** Reduzir o número de casos de teste necessários, garantindo que cada classe de equivalência seja testada ao menos uma vez.
  - **Exemplo:** Notas válidas (0 a 10), notas inválidas (menores que 0, maiores que 10).
- **Análise de Valor Limite:**
  - **Critério:** Focar nos limites extremos dos intervalos de entrada, onde os erros são mais prováveis.
  - **Finalidade:** Detectar defeitos em pontos críticos, como exatamente 0, 6, 10, 12 pontos totais.
  - **Exemplo:** Testar notas exatamente iguais a 6 (nota mínima) e totais exatamente iguais a 12 (mínimo para aprovação).
- **Teste de Cenários:**
  - **Critério:** Criar situações que simulam fluxos reais de uso, combinando várias condições e entradas.
  - **Finalidade:** Avaliar como o sistema se comporta em situações complexas e próximas do uso real.
  - **Exemplo:** Aluno com uma nota alta e outra baixa, totalizando mais de 12 pontos, mas com uma nota abaixo de 6.

2.2. Teste Estrutural (Caixa Branca)

Objetivo: Avaliar a estrutura interna do código-fonte, garantindo que todos os caminhos possíveis de execução sejam testados.

Técnicas Aplicadas:

- **Cobertura de Declarações:**
  - **Critério:** Garantir que todas as linhas de código sejam executadas ao menos uma vez durante os testes.
  - **Finalidade:** Identificar código não utilizado ou inacessível.
  - **Exemplo:** Verificar que todas as instruções de validação de notas e cálculo de status são executadas.
- **Cobertura de Ramos (Decisões):**
  - **Critério:** Garantir que cada decisão no código (if, else) seja avaliada tanto como verdadeira quanto falsa.
  - **Finalidade:** Testar todos os caminhos lógicos do programa.
  - **Exemplo:** Testar cenários onde o aluno é aprovado, reprovado ou precisa de prova extra.

2.3. Teste de Mutação

Objetivo: Avaliar a qualidade e eficácia dos casos de teste através da introdução de pequenas alterações (mutações) no código e verificar se os casos de teste conseguem detectar essas alterações.

Critérios Utilizados:

- **Geração de Mutantes:**
  - Alteração de operadores aritméticos (e.g., trocar `+` por `-`).
  - Modificação de operadores relacionais (e.g., trocar `>=` por `>`).
  - Substituição de valores constantes (e.g., alterar o valor mínimo de aprovação de 12 para outro número).
- **Detecção de Mutantes:**
  - Executar os casos de teste contra os mutantes.
  - Verificar se os testes falham quando executados no código mutado.

Finalidade:

- **Avaliar Cobertura dos Testes:** Certificar-se de que os casos de teste são robustos o suficiente para capturar erros introduzidos.
- **Melhorar Casos de Teste:** Identificar áreas onde os testes podem ser aprimorados para aumentar a detecção de defeitos.

Ferramentas Utilizadas:

- **Frameworks de Teste Unitário e Ferramente de cobertura de Código:**
  - **Jest:** Framework de teste em JavaScript que permite escrever testes claros e manter o código organizado.

        - **Objetivo:** Facilitar a escrita e execução de testes unitários e de integração. Também medir quais partes do código estão sendo executadas durante os testes.
- **Ferramentas de Teste de Mutação:**
    - **Stryker Mutator:** Ferramenta para realizar testes de mutação em projetos JavaScript.
        - **Objetivo:** Automatizar o processo de geração de mutantes e avaliar a eficácia dos casos de teste.
- **Análise Estática de Código:**
    - **ESLint:** Ferramenta para identificar problemas no código JavaScript.
        - **Objetivo:** Encontrar erros comuns e manter boas práticas de codificação.
- **Ambiente de Desenvolvimento:**
    - **Visual Studio Code:** IDE com suporte a extensões para facilitar o desenvolvimento e teste.
        - **Objetivo:** Proporcionar um ambiente integrado para escrever, depurar e testar o código.

3. Projeto de Casos de Teste (como será testado)

Teste Funcional (Caixa Preta)

Técnica: Particionamento de Equivalência

Variáveis de Entrada:

- **Nota1**: primeira nota do aluno (esperado entre 0 e 10).
- **Nota2**: segunda nota do aluno (esperado entre 0 e 10).

Classes de Equivalência para cada variável:

- **Classes Válidas:**
    - **V1**: Notas entre 0 e 10 (inclusive).
- **Classes Inválidas:**
    - **I1**: Notas menores que 0.
    - **I2**: Notas maiores que 10.
    - **I3**: Notas não numéricas (e.g., letras, símbolos).
    - **I4**: Notas nulas ou indefinidas.

Casos de Teste Baseados no Particionamento de Equivalência:

Nota1	Classe de Nota1	Nota2	Classe de Nota2	Resultado Esperado
8	V1	7	V1	Aprovado sem necessidade de prova extra
-1	I1	5	V1	Erro na entrada da Nota1 (valor negativo)
6	V1	11	I2	Erro na entrada da Nota2 (valor acima de 10)
'A'	I3	9	V1	**Erro**: Entrada inválida para Nota1
null	I4	5	V1	**Erro**: Entrada inválida para Nota1
5	V1	7	V1	Resultado: Necessita de prova extra.
6	V1	5	V1	Reprovado
5.5	V1	6.5	V1	Necessita de prova extra

Técnica: Análise de Valor Limite

Limites das Variáveis de Entrada:

- **Nota1 e Nota2:**
  - **Limite Inferior:** 0
  - **Limite Superior:** 10
- **Nota Mínima por Prova:** 6
- **Total Mínimo para Aprovação:** 12 pontos

Casos de Teste Baseados na Análise de Valor Limite:

Nota1	Nota2	Total	Resultado Esperado
0	0	0	Reprovado
10	10	20	Aprovado, não necessita de prova extra
6	6	12	Aprovado, não necessita de prova extra
5.9	6	11.9	Reprovado
6	5.9	11.9	Reprovado
5.9	6.1	12	Necessita de prova extra
6	5	11	Reprovado
6	6	12	Aprovado, não necessita de prova extra

Teste Estrutural (Caixa Branca)

Critério: Cobertura de Declarações e Ramos

Objetivo: Garantir que todas as linhas de código e condições lógicas sejam executadas pelos casos de teste.

Análise dos Caminhos Lógicos:

1. **Validação das Notas:**
   - Notas dentro do intervalo válido (0 a 10).
   - Notas fora do intervalo (menores que 0 ou maiores que 10).
   - Notas não numéricas ou nulas.
2. **Cálculo do Total:**
   - Total ≥ 12 pontos.
   - Total < 12 pontos.
3. **Verificação das Notas Individuais:**
   - Nota ≥ 6.
   - Nota < 6.
4. **Determinação do Resultado Final:**
   - Aprovado sem prova extra.
   - Aprovado com necessidade de prova extra.
   - Reprovado.

Casos de Teste para Detectar Mutantes

Após os primeiros testes, foram encontrados inicialmente 3 mutantes, mas criando novos testes eles foram eliminados.

Tabela Final Consolidada de Casos de Teste

Caso de Teste	Técnica Aplicada	Nota1	Nota2	Total	Resultado Esperado
1	Funcional - Equivalência	8	7	15	Aprovado, não necessita de prova extra.
2	Funcional - Equivalência	5	7	12	Necessita de prova extra.
3	Funcional - Valor Limite	4	7	11	Reprovado

Caso de Teste	Técnica Aplicada	Nota1	Nota2	Total	Resultado Esperado
4	Funcional - Valor Limite	-1	5	N/A	Erro: Nota1 inválida. Insira um número entre 0 e 10.
5	Funcional - Valor Limite	6	11	N/A	Erro: Nota2 inválida. Insira um número entre 0 e 10.
6	Estrutural - Cobertura de Ramos	6	6	12	Aprovado, não necessita de prova extra.
7	Estrutural - Cobertura de Ramos	5.9	6.1	12	Necessita de prova extra.
8	Funcional - Equivalência	0	0	0	Reprovado.
9	Estrutural	'A'	6	N/A	Erro: Nota1 inválida. Insira um número entre 0 e 10.
10	Estrutural	6	'B'	N/A	Erro: Nota2 inválida. Insira um número entre 0 e 10.
11	Teste de Mutação (Mutante 1)	'A'	7	N/A	Erro: Nota1 inválida. Insira um número entre 0 e 10.
12	Teste de Mutação (Mutante 1)	6	null	N/A	Erro: Nota2 inválida. Insira um número entre 0 e 10.
13	Teste de Mutação (Mutante 1)	undefined	5	N/A	Erro: Nota1 inválida. Insira um número entre 0 e 10.
14	Teste de Mutação (Mutante 2)	10	10	20	Aprovado, não necessita de prova extra.
15	Teste de Mutação (Mutante 2)	10	7	17	Aprovado, não necessita de prova extra.
16	Teste de Mutação (Mutante 3)	7	5	12	Necessita de prova extra.
17	Teste de Mutação (Mutante 3)	6	5.9	11.9	Reprovado.

---

4. Execução (quando e como será testado)

Detalhes sobre a Execução dos Testes:

Os testes foram executados após a implementação do módulo de cálculo de notas acadêmicas em JavaScript ter sido concluída. O processo de teste ocorreu nas seguintes etapas:

1. **Preparação do Ambiente de Teste:**
   - **Configuração do Ambiente:** O ambiente de desenvolvimento foi configurado utilizando o **Node.js** para executar o código JavaScript em ambiente de servidor.
   - **Instalação de Ferramentas:** As seguintes ferramentas foram instaladas para auxiliar na execução e automação dos testes:
     - **Jest:** Framework de teste JavaScript para executar testes unitários e de integração.
     - **Stryker Mutator:** Ferramenta para realizar testes de mutação e avaliar a eficácia dos casos de teste.
     - **ESLint:** Utilizado para análise estática do código e detecção de erros de sintaxe ou más práticas.
2. **Implementação dos Casos de Teste:**
   - **Criação dos Arquivos de Teste:** Para cada caso de teste definido anteriormente, foram criados scripts de teste utilizando o Jest.
   - **Organização dos Testes:** Os testes foram organizados em uma suite, agrupando casos similares, como testes de validação de entrada, cálculo do total de pontos e determinação do status do aluno.
3. **Execução dos Testes:**
   - **Execução dos Testes Unitários e de Cobertura:** Os testes foram executados utilizando o comando `jest` no terminal, o que permitiu verificar individualmente cada função do módulo. Foi também utilizado para gerar relatórios detalhados sobre a cobertura de código, garantindo que todas as partes do código foram testadas.
   - **Execução dos Testes de Mutação:**
     - O Stryker Mutator foi configurado para gerar mutantes do código.
     - Os casos de teste foram executados contra os mutantes para verificar se eram capazes de detectar as alterações introduzidas.
     - Relatórios de mutação foram gerados para avaliar a porcentagem de mutantes detectados.
4. **Análise dos Resultados:**
   - **Comparação dos Resultados Obtidos com os Esperados:** Para cada caso de teste, o resultado retornado pelo sistema foi comparado com o resultado esperado definido anteriormente.
   - **Registro de Falhas:** Qualquer discrepância entre o resultado obtido e o esperado foi registrada, incluindo detalhes sobre o erro ocorrido.

- **Atualização dos Casos de Teste:** Em casos onde os testes não detectaram erros em mutantes, os casos de teste foram revisados e aprimorados para aumentar sua eficácia.

Resultados dos Casos de Teste:
Os resultados do caso de teste foram todos de acordo com a tabela consolidada de casos de teste.

Descrição dos Erros Encontrados:

Durante a execução dos testes, foram identificados alguns erros que foram registrados para correção:

Erro 1: Validação Inadequada de Notas Não Numéricas

- **Casos de Teste Afetados:**
  - Caso de Teste 4 (Nota1: 'A', Nota2: 9)
- **Descrição do Erro:**
  - O sistema não estava validando corretamente entradas não numéricas, permitindo que valores como strings fossem processados.
- **Resultado Obtido:**
  - O sistema retornou um erro ou comportamento inesperado.
- **Ação Tomada:**
  - O código foi revisado para incluir validação adicional que verifica se as notas são números.

Erro 2: Problemas com Notas Decimais

- **Casos de Teste Afetados:**
  - Caso de Teste 14 (Nota1: 5.9, Nota2: 6.1)
- **Descrição do Erro:**
  - O sistema estava arredondando as notas decimais, resultando em cálculos incorretos do total de pontos.
- **Resultado Obtido:**
  - O total calculado não correspondia ao esperado.
- **Ação Tomada:**
  - O código foi ajustado para trabalhar corretamente com números decimais, utilizando precisão adequada nas operações aritméticas.

Erro 3: Falha na Detecção de Mutantes

- **Casos de Teste Afetados:**
  - Durante os testes de mutação com o Stryker, alguns mutantes não foram detectados pelos casos de teste existentes.
- **Descrição do Erro:**
  - Os casos de teste não cobriam certos cenários necessários para detectar mutações específicas, indicando falta de cobertura.
- **Ação Tomada:**
  - Novos casos de teste foram adicionados para cobrir os cenários faltantes, aumentando a eficácia dos testes.

Ferramentas Utilizadas e Como Foram Aplicadas:

- **Jest:**
  - Utilizado para escrever e executar testes unitários, além de ser utilizado para medir a cobertura de código.
  - Permitiu a criação de testes claros e organizados para cada função do módulo.
  - Forneceu feedback imediato sobre o sucesso ou falha dos testes.
  - Ajudou a identificar áreas não testadas que poderiam conter defeitos.
- **Stryker Mutator:**
  - Configurado para realizar testes de mutação no código JavaScript.
  - Gerou mutantes automaticamente, modificando pequenas partes do código.
  - Avaliou a capacidade dos casos de teste em detectar essas mutações.
  - Relatórios indicaram o percentual de mutantes "mortos" (detectados) e "sobreviventes" (não detectados).
- **ESLint:**
  - Realizou análise estática do código.
  - Detectou erros de sintaxe, más práticas e problemas potenciais antes da execução dos testes.
  - Garantiu que o código seguisse padrões consistentes de estilo e qualidade.

5. Análise dos Resultados e próximos passos

Após a execução dos testes planejados para o sistema de cálculo de notas acadêmicas, várias conclusões importantes foram alcançadas:

- **Funcionamento Conforme Especificado:**
  - Os testes funcionais confirmaram que o sistema atende aos requisitos especificados, calculando corretamente o total de pontos e determinando o status do aluno (aprovado, reprovado ou necessidade de prova extra) com base nas notas inseridas.

    - As validações de entrada estão funcionando adequadamente, rejeitando valores inválidos e solicitando novas entradas quando necessário.
- **Cobertura de Código e Eficiência dos Casos de Teste:**
    - A cobertura de código atingiu um nível satisfatório, com todos os caminhos lógicos principais sendo testados.
    - Os testes estruturais garantiram que todas as declarações e condições foram executadas, aumentando a confiabilidade do sistema.
    - Os testes de mutação mostraram que os casos de teste são eficazes em detectar alterações sutis no código, embora alguns mutantes tenham sobrevivido inicialmente, indicando áreas para melhoria nos testes.
- **Detecção e Correção de Defeitos:**
    - Foram identificados e corrigidos defeitos relacionados à validação de entradas não numéricas e ao tratamento de números decimais.
    - A execução iterativa dos testes permitiu aprimorar o código e fortalecer as verificações de entrada.

Dada a possibilidade de dispender mais tempo na realização dos testes, os seguintes passos poderiam ser considerados para aprimorar ainda mais o sistema:

Teste de Usabilidade e Feedback de Usuários Reais:

- **Sessões de Teste com Usuários Finais:**
    - Obter feedback direto de professores, alunos ou outros usuários potenciais.
    - **Benefício:** Identificar problemas de usabilidade ou funcionalidades adicionais que poderiam ser incorporadas.
- **Melhorias na Interface do Usuário:**
    - Construir uma interface para tornar o sistema mais intuitivo e amigável.