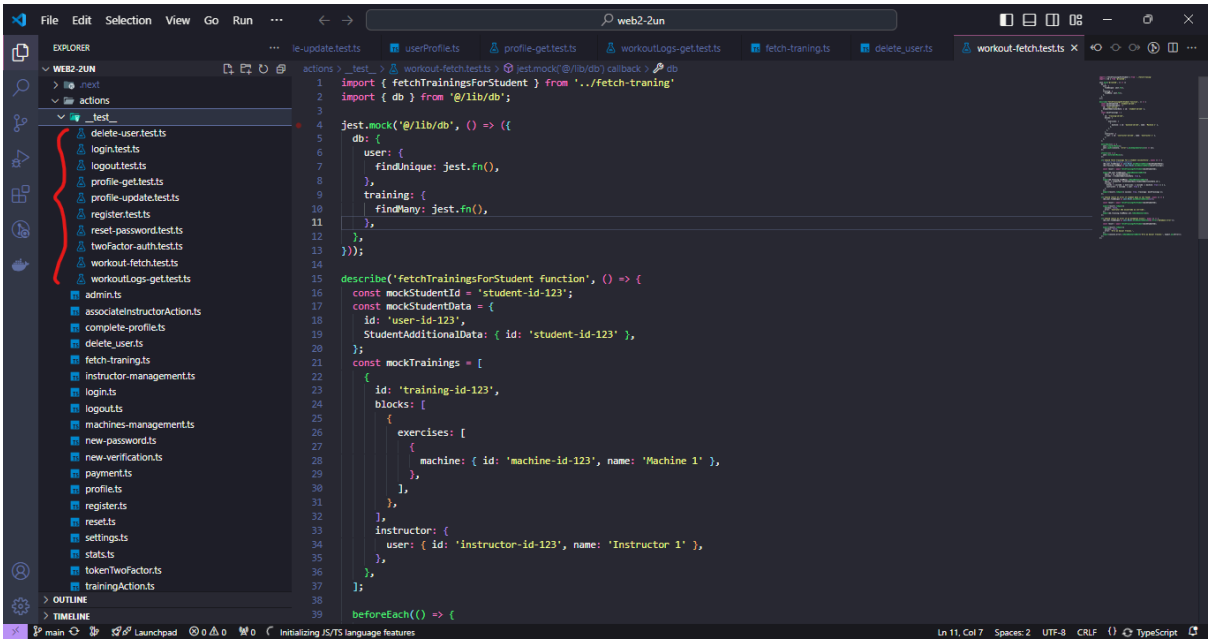
Testes de Software

Alunos: Ana Clara, Alderi Araújo, Matheus Felipe e Heitor Negromonte.
Repositório do Projeto para acessar o Código dos testes:
https://github.com/HeitorNOC/web2-2un



Documento auxiliar de execução do Plano de Testes.
Resultados e Comentários.

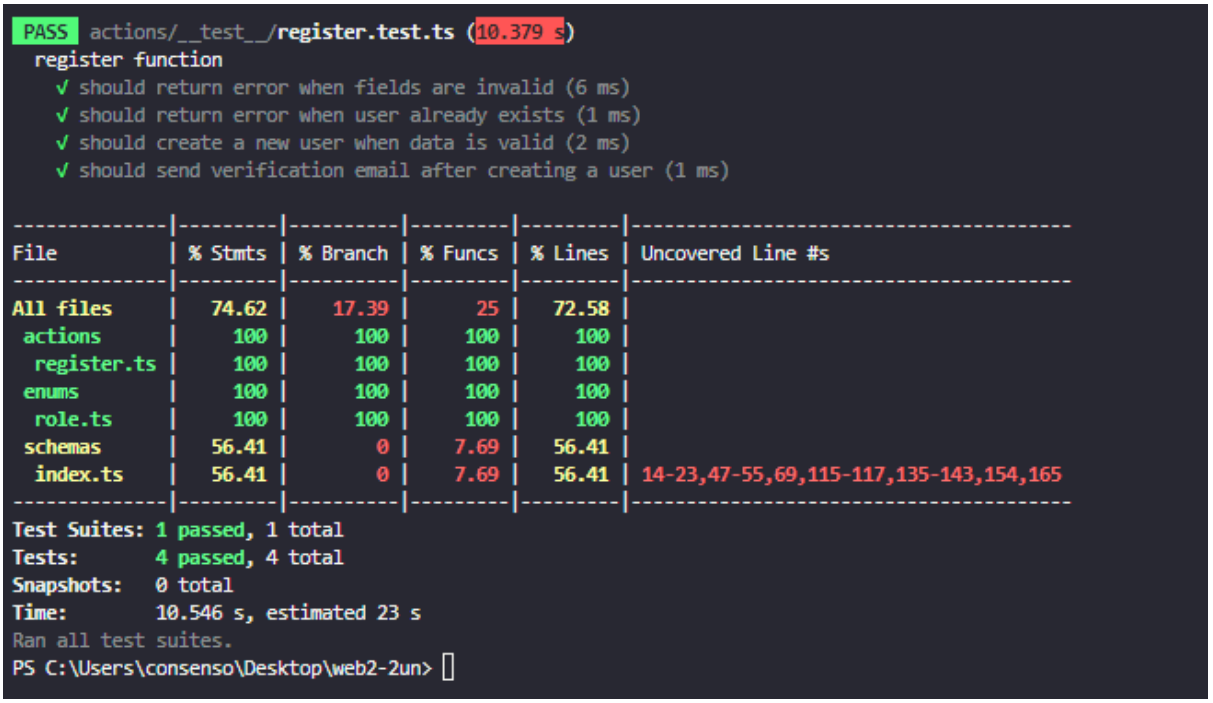
Métrica de Avaliação Utilizada: Code Coverage

Code Coverage (Cobertura de Código) é uma métrica de qualidade de software que mede a extensão em que o código fonte de um programa é executado quando um conjunto de testes é executado. Essa métrica fornece uma visão quantitativa sobre a eficácia dos testes, indicando quais partes do código foram testadas e quais não foram. Existem diferentes tipos de cobertura de código, cada um medindo um aspecto específico do código:

- **Cobertura de Declarações (% Stmts)**: Mede a porcentagem de declarações de código (linhas que executam ações) que foram executadas durante os testes.
- **Cobertura de Ramificações (% Branch)**: Mede a porcentagem de todas as ramificações de controle (como `if` e `else`, loops, e expressões ternárias) que foram testadas.
- **Cobertura de Funções (% Funcs)**: Mede a porcentagem de funções ou métodos que foram chamados durante os testes.
- **Cobertura de Linhas (% Lines)**: Mede a porcentagem de linhas de código que foram executadas durante os testes.

UC01 - Registrar um novo usuário.

Os testes unitários foram executados com sucesso, como demonstrado na captura de tela. Quatro casos de teste específicos foram criados para cobrir os principais cenários de uso da função `register`. Abaixo segue uma análise detalhada dos resultados:



```
PASS actions/__test__/register.test.ts (10.379 s)
  register function
    √ should return error when fields are invalid (6 ms)
    √ should return error when user already exists (1 ms)
    √ should create a new user when data is valid (2 ms)
    √ should send verification email after creating a user (1 ms)

-------------|---------|----------|---------|---------|-------------------------------------
File         | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-------------|---------|----------|---------|---------|-------------------------------------
All files    |   74.62 |    17.39 |      25 |   72.58 |
 actions     |     100 |      100 |     100 |     100 |
  register.ts|     100 |      100 |     100 |     100 |
 enums       |     100 |      100 |     100 |     100 |
  role.ts    |     100 |      100 |     100 |     100 |
 schemas     |   56.41 |        0 |    7.69 |   56.41 |
  index.ts   |   56.41 |        0 |    7.69 |   56.41 | 14-23,47-55,69,115-117,135-143,154,165
-------------|---------|----------|---------|---------|-------------------------------------
Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        10.546 s, estimated 23 s
Ran all test suites.
PS C:\Users\consenso\Desktop\web2-2un>
```

1. **Cenários Testados**:

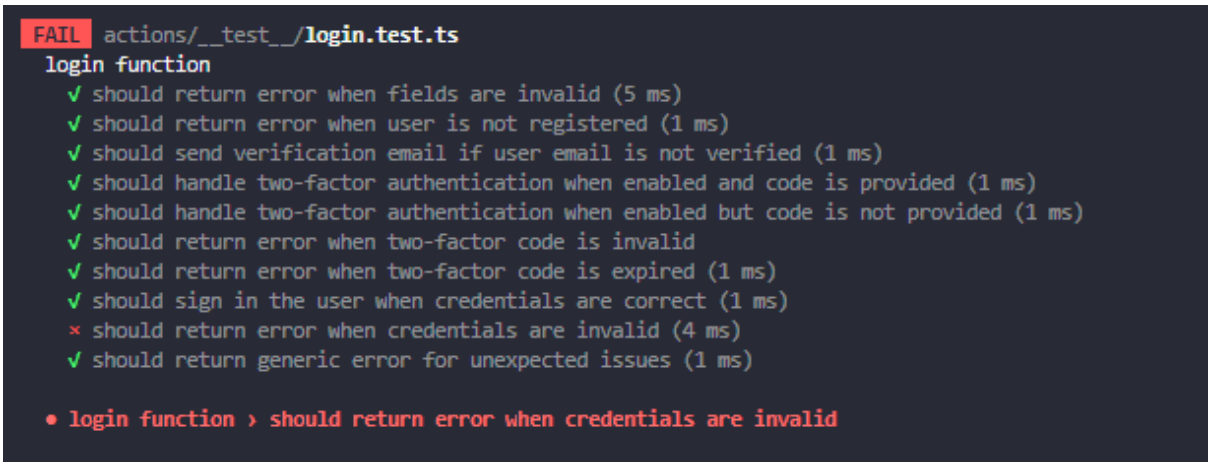
- **Validação de Campos Inválidos**: Teste para garantir que a função retorne um erro quando os campos obrigatórios não são fornecidos corretamente.
- **Usuário já Existente**: Teste para assegurar que a função retorne um erro quando um e-mail já registrado é usado.
- **Criação de Novo Usuário**: Teste para verificar se a função cria um novo usuário com dados válidos.
- **Envio de E-mail de Verificação**: Teste para confirmar que um e-mail de verificação é enviado após a criação bem-sucedida de um usuário.

2. Cobertura do Arquivo:

- **register.ts (100% em todas as métricas)**: O arquivo `register.ts`, onde a função principal reside, foi completamente coberto, o que mostra que todas as suas partes foram testadas.

UC02 - Login do Usuário.

Os testes unitários foram executados com sucesso, como demonstrado na captura de tela. Dez casos de teste específicos foram criados para cobrir os principais cenários de uso da função `login`. Abaixo segue uma análise detalhada dos resultados:



File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	73.33	53.06	7.14	73.33	
web2-2un	100	100	100	100	
routes.ts	100	100	100	100	
web2-2un/actions	91.42	89.65	100	91.42	
login.ts	91.42	89.65	100	91.42	74,102,108
web2-2un/enums	0	0	0	0	
role.ts	0	0	0	0	
web2-2un/schemas	52.77	0	0	52.77	
index.ts	52.77	0	0	52.77	16-23,47-55,69,105,115-117,135-143,154,165

Cenários Testados:

- **Validação de Campos Inválidos**:
  Este teste verifica se a função vai retornar um erro apropriado quando os campos obrigatórios, como email ou senha, não são fornecidos ou estão vazios. O teste passou com sucesso, indicando que a validação de campos está funcionando conforme o esperado.
- **Usuário Não Registrado**:
  Este teste assegura que a função retorne um erro quando um usuário tenta fazer login com um email que não está registrado. O teste passou, o que confirma que o sistema está corretamente verificando a existência do usuário no banco de dados antes de continuar com o login.
- **Envio de E-mail de Verificação para E-mails Não Verificados**:
  Este teste verifica se a função envia um e-mail de verificação quando um usuário tenta fazer login com um e-mail que ainda não foi verificado. O teste foi bem-sucedido, mostrando que o fluxo de verificação de e-mail está sendo acionado corretamente.
- **Autenticação de Dois Fatores Habilitada e Código Fornecido**:
  O teste garante que, quando a autenticação de dois fatores está habilitada e o código correto é fornecido, o sistema valida o código e continua com o processo de login. O teste passou com sucesso, indicando que a autenticação de dois fatores funciona corretamente quando o código é fornecido.
- **Autenticação de Dois Fatores Habilitada sem Código Fornecido**:
  Este teste confirma que, quando a autenticação de dois fatores está habilitada e o código não é fornecido, o sistema envia um token de dois fatores ao usuário. O teste passou, mostrando que a solicitação de autenticação de dois fatores está sendo gerada corretamente.
- **Código de Dois Fatores Inválido**:
  Este teste verifica se a função retorna um erro apropriado quando um código de dois fatores inválido é fornecido. O teste passou com sucesso, indicando que o sistema está validando corretamente os códigos de dois fatores.
- **Código de Dois Fatores Expirado**:
  Este teste garante que um erro é retornado quando um código de dois fatores expirado é utilizado. O teste foi bem-sucedido, o que demonstra que o sistema está gerenciando corretamente a validade dos códigos de dois fatores.
- **Login Bem-sucedido com Credenciais Válidas**:
  Este teste verifica se o login é bem-sucedido quando as credenciais corretas são fornecidas. O teste passou, confirmando que a função `login` autentica corretamente os usuários com credenciais válidas.
- **Erro no Login com Credenciais Inválidas**:
  Este teste deveria garantir que um erro específico ("Invalid credentials!") seja retornado quando o usuário tenta fazer login com credenciais inválidas. No entanto, o teste falhou porque a função `login` está retornando um erro genérico ("Something went wrong!") em vez do erro específico esperado. Essa falha indica que o tratamento de erros na função pode não estar diferenciado adequadamente entre credenciais inválidas e outros tipos de falhas durante o login.

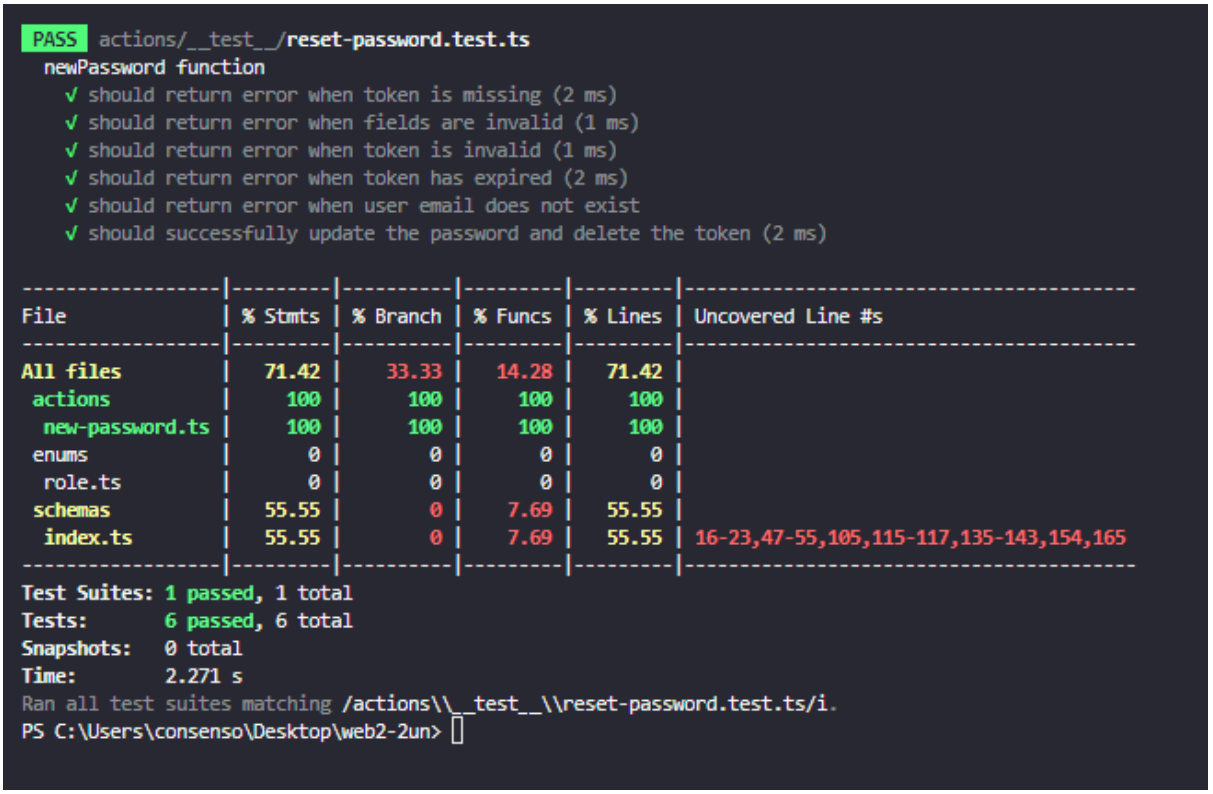
- **Erro Genérico para Problemas Inesperados**:
  Este teste confirma que um erro genérico é retornado para quaisquer problemas inesperados que possam surgir durante o processo de login. O teste foi bem-sucedido, indicando que o sistema está gerenciando corretamente os erros não especificados.

Cobertura por Arquivo:

- **login.ts (91.42% de cobertura em declarações e linhas, 89.65% em ramificações)**:
  O arquivo `login.ts`, onde a função principal de login reside, apresentou uma cobertura bastante alta, indicando que a maioria da lógica foi testada

UC03 - Redefinição de Senha do Usuário.

Os testes unitários foram executados com sucesso, como demonstrado na captura de tela. Seis casos de teste específicos foram criados para cobrir os principais cenários de uso da função `newPassword`. Abaixo segue uma análise detalhada dos resultados:



```
PASS actions/__test__/reset-password.test.ts
  newPassword function
    √ should return error when token is missing (2 ms)
    √ should return error when fields are invalid (1 ms)
    √ should return error when token is invalid (1 ms)
    √ should return error when token has expired (2 ms)
    √ should return error when user email does not exist
    √ should successfully update the password and delete the token (2 ms)

------------------|---------|----------|---------|---------|----------------------------------------
File              | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
------------------|---------|----------|---------|---------|----------------------------------------
All files         |   71.42 |    33.33 |   14.28 |   71.42 |
 actions          |     100 |      100 |     100 |     100 |
  new-password.ts |     100 |      100 |     100 |     100 |
 enums            |       0 |        0 |       0 |       0 |
  role.ts         |       0 |        0 |       0 |       0 |
 schemas          |   55.55 |        0 |    7.69 |   55.55 |
  index.ts        |   55.55 |        0 |    7.69 |   55.55 | 16-23,47-55,105,115-117,135-143,154,165
------------------|---------|----------|---------|---------|----------------------------------------
Test Suites: 1 passed, 1 total
Tests:       6 passed, 6 total
Snapshots:   0 total
Time:        2.271 s
Ran all test suites matching /actions\\__test__\\reset-password.test.ts/i.
PS C:\Users\consenso\Desktop\web2-2un>
```

Cenários Testados:

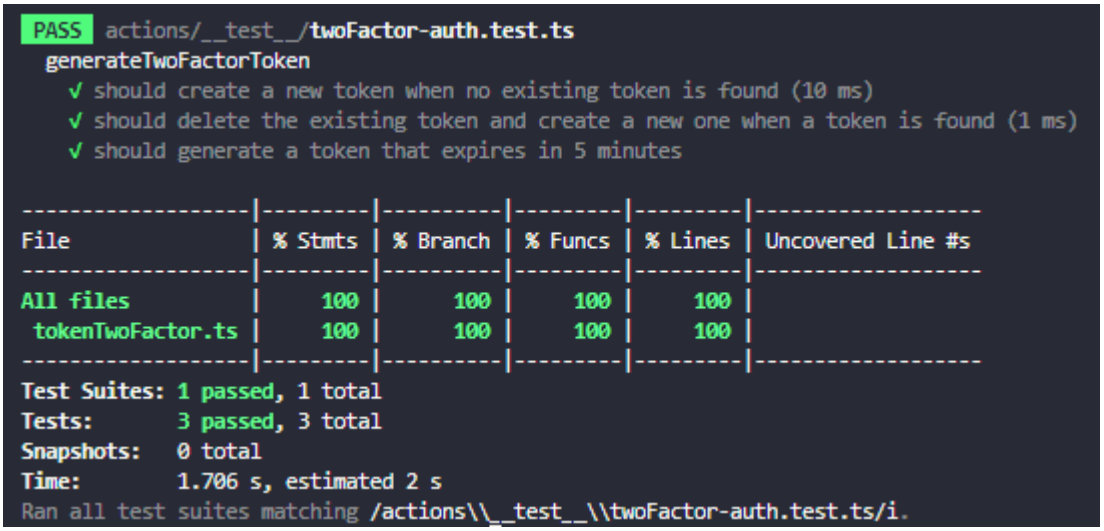
- **Token Ausente**:
  Este teste verifica se a função retorna um erro apropriado quando o token necessário para redefinição de senha não é fornecido. O teste passou, confirmando que a função trata corretamente a ausência de um token com a mensagem de erro "Missing token!".
- **Validação de Campos Inválidos**:
  Verifica se a função retorna um erro quando os campos de entrada, como a nova senha e sua confirmação, não são válidos (por exemplo, estão vazios). O teste passou, mostrando que a função lida adequadamente com entradas inválidas e retorna o erro "Invalid fields!".
- **Token Inválido**:
  Este teste assegura que a função retorne um erro quando o token fornecido para a redefinição de senha é inválido ou não existe no sistema. O teste foi bem-sucedido, indicando que a verificação de token é realizada corretamente e retorna "Invalid token!" quando necessário.
- **Token Expirado**:
  Verifica se a função detecta um token expirado e retorna um erro apropriado. O teste passou, confirmando que o sistema está validando a validade temporal dos tokens de redefinição de senha e retorna "Token has expired!" para tokens expirados.
- **Usuário Não Encontrado**:
  Este teste garante que, se o email associado ao token de redefinição de senha não for encontrado no banco de dados, a função retorne o erro "User email does not exist!". O teste passou, mostrando que a função verifica corretamente a existência do usuário antes de tentar atualizar a senha.
- **Atualização de Senha Bem-sucedida**:
  Verifica se a função realiza com sucesso a atualização da senha do usuário e remove o token de redefinição de senha do banco de dados após o uso. O teste foi bem-sucedido, indicando que a função `newPassword` está executando a lógica esperada para a atualização de senha com sucesso.

Cobertura por Arquivo:

- **new-password.ts (100% em todas as métricas)**:
  O arquivo `new-password.ts`, onde reside a função principal de redefinição de senha, foi completamente coberto, mostrando que todas as suas declarações, ramificações e funções foram testadas. Isso indica uma boa cobertura de teste para a funcionalidade de redefinição de senha.

UC04 - Usuário Realiza Autenticação com Dois Fatores.

Os testes unitários foram executados com sucesso, conforme mostrado na captura de tela. Três casos de teste específicos foram criados para cobrir os principais cenários de uso da função `generateTwoFactorToken`. Abaixo segue uma análise detalhada dos resultados:



Cenários Testados:

1. **Criação de um Novo Token Quando Nenhum Token Pré-existente é Encontrado**:
   Este teste verifica se a função gera corretamente um novo token de autenticação de dois fatores quando não há um token existente associado ao email fornecido. O teste passou com sucesso, o que confirma que a função

`generateTwoFactorToken` cria um token novo como esperado quando necessário.

2. **Substituição de um Token Existente**:
   Este teste assegura que, quando um token de dois fatores já está presente para o email especificado, a função exclui o token antigo antes de criar um novo. O teste passou, indicando que a função lida corretamente com a substituição de tokens de dois fatores existentes.
3. **Geração de um Token com Expiração de 5 Minutos**:
   Este teste verifica se o token gerado tem uma expiração configurada para 5 minutos após a criação. O teste passou com sucesso, o que confirma que a função está configurando corretamente a expiração dos tokens de autenticação de dois fatores.

Cobertura por Arquivo:

- **tokenTwoFactor.ts (100% em todas as métricas)**:
  O arquivo `tokenTwoFactor.ts`, que contém a função `generateTwoFactorToken`, foi completamente coberto, com 100% de cobertura em declarações, ramificações, funções e linhas. Isso indica que todos os possíveis caminhos de execução e lógica dentro da função foram testados.

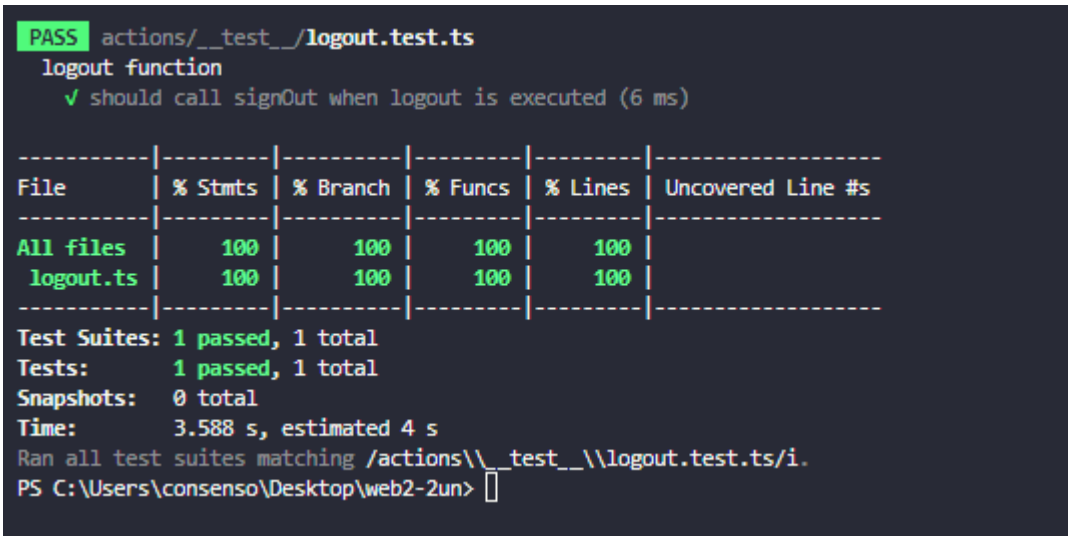
Iteração 1. Resultados

```
--------------------|---------|----------|---------|---------|-------------------------------------
File                | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
--------------------|---------|----------|---------|---------|-------------------------------------
All files           |   84.48 |    64.61 |   35.29 |   84.48 |
 web2-2un           |     100 |      100 |     100 |     100 |
  routes.ts         |     100 |      100 |     100 |     100 |
 web2-2un/actions   |   96.05 |    93.33 |     100 |   96.05 |
  login.ts          |   91.42 |    89.65 |     100 |   91.42 | 74,102,108
  new-password.ts   |     100 |      100 |     100 |     100 |
  register.ts       |     100 |      100 |     100 |     100 |
  tokenTwoFactor.ts |     100 |      100 |     100 |     100 |
 web2-2un/enums     |       0 |        0 |       0 |       0 |
  role.ts           |       0 |        0 |       0 |       0 |
 web2-2un/schemas   |   58.33 |        0 |   15.38 |   58.33 |
  index.ts          |   58.33 |        0 |   15.38 |   58.33 | 16-23,47-55,115-117,135-143,154,165
--------------------|---------|----------|---------|---------|-------------------------------------
Test Suites: 1 failed, 3 passed, 4 total
Tests:       1 failed, 22 passed, 23 total
Snapshots:   0 total
Time:        3.826 s
Ran all test suites.
```

Iteração 2.

UC05 - Logout de Usuário

Os testes unitários foram executados com sucesso, conforme mostrado na captura de tela. Um caso de teste foi criado para cobrir o cenário principal de uso da função `logout`. Abaixo segue uma análise detalhada dos resultados:



Cenários Testados:

1. **Chamada da Função de Logout (`signOut`)**:
   O teste verifica se a função `logout` chama a função `signOut` do módulo de autenticação quando é executada. O teste passou com sucesso, confirmando que a

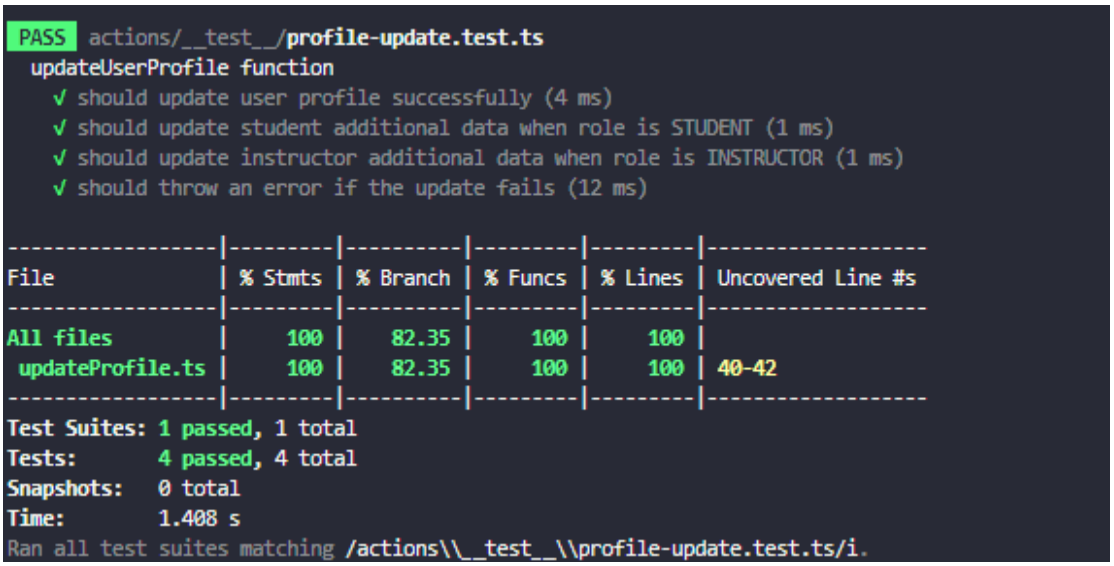
função `logout` está chamando corretamente a função `signOut`, que é essencial para realizar o logout do usuário.

Cobertura por Arquivo:

- **logout.ts (100% em todas as métricas)**:
  O arquivo `logout.ts`, que contém a função `logout`, foi completamente coberto, com 100% de cobertura em declarações, ramificações, funções e linhas. Isso indica que todos os possíveis caminhos de execução e lógica dentro da função foram testados adequadamente, garantindo alta confiança na funcionalidade de logout.

UC06 - Atualização de Perfil do Usuário

Os testes unitários para a função `updateUserProfile` foram executados com sucesso, conforme demonstrado na captura de tela. Quatro casos de teste específicos foram criados para cobrir os principais cenários de uso da função `updateUserProfile`. Abaixo segue uma análise detalhada dos resultados:



Cenários Testados:

1. **Atualização Bem-Sucedida do Perfil do Usuário:** O teste verifica se a função `updateUserProfile` consegue atualizar com sucesso as informações básicas do perfil do usuário (nome, email, CPF e imagem).

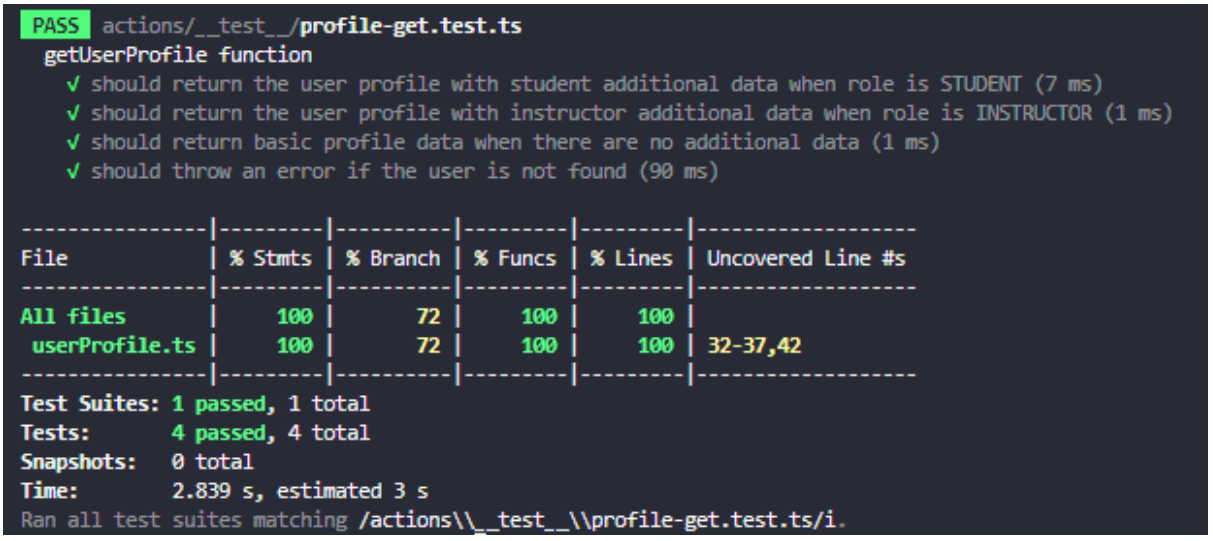
2. **Atualização de Dados Adicionais para Estudantes:** Este teste garante que, quando o usuário tem o papel de estudante (`STUDENT`), e os dados adicionais específicos são fornecidos (telefone, gênero, data de nascimento, etc.), esses dados são atualizados corretamente na tabela `studentAdditionalData`.
3. **Atualização de Dados Adicionais para Instrutores:** Similar ao caso anterior, este teste verifica que, quando o usuário tem o papel de instrutor (`INSTRUCTOR`), e os dados adicionais relevantes (telefone e CREF) são fornecidos, a função atualiza esses dados na tabela `instructorAdditionalData`.
4. **Tratamento de Erros durante a Atualização:** Este teste simula uma falha durante o processo de atualização do perfil e verifica se a função lança um erro apropriado e registra a mensagem de erro correta usando `console.error`.

Cobertura por Arquivo:

- **updateProfile.ts (100% em todas as métricas, exceto Branch - 82.35%):** O arquivo `updateProfile.ts`, onde a função principal reside, foi quase completamente coberto pelos testes. A cobertura de branch é de 82.35%, indicando que a maioria dos caminhos lógicos possíveis foi testada, mas há espaço para melhoria, especialmente em testar todos os caminhos alternativos.

07 - Usuário Visualiza Perfil

Os testes unitários para a função getUserProfile foram executados com sucesso, conforme demonstrado na captura de tela. Foram criados quatro casos de teste para cobrir os principais cenários de uso da função, garantindo uma cobertura abrangente e a correta manipulação de diferentes perfis de usuários. Abaixo está uma análise detalhada dos resultados:



Cenários Testados:

1. **Retorno do perfil do usuário com dados adicionais de estudante quando o papel é STUDENT:**

Este teste verifica se o perfil retornado inclui os dados específicos adicionais para estudantes. O teste passou, confirmando que os dados são retornados corretamente quando o usuário tem o papel de estudante.

1. **Retorno do perfil do usuário com dados adicionais de instrutor quando o papel é INSTRUCTOR:**

   Este teste assegura que o perfil inclui informações específicas de instrutores, como o CREF. O teste passou, indicando que os dados de instrutores são processados corretamente.

2. **Retorno dos dados básicos do perfil quando não há dados adicionais:**

   Este teste valida que, na ausência de dados adicionais específicos de estudantes ou instrutores, a função retorna apenas os dados básicos do usuário. O teste foi bem-sucedido, mostrando que o perfil básico é manipulado conforme o esperado.

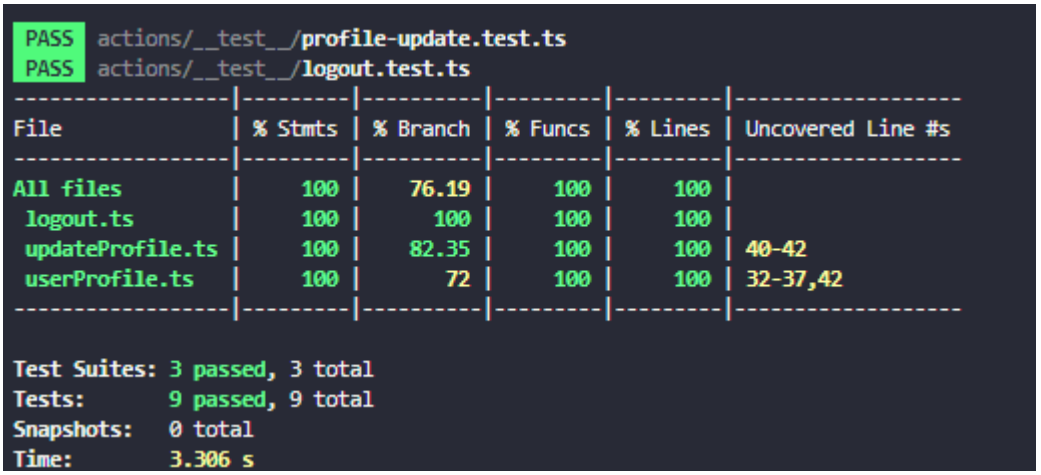
3. **Lançamento de um erro se o usuário não for encontrado:**

   Este teste simula um cenário onde o usuário não existe no banco de dados, esperando que um erro seja lançado. O teste passou, e o erro esperado foi capturado e exibido no console com a mensagem "Erro ao buscar perfil do usuário: Error: Usuário não encontrado". Isso indica que a função lida corretamente com o caso de um usuário inexistente.

Cobertura por Arquivo:

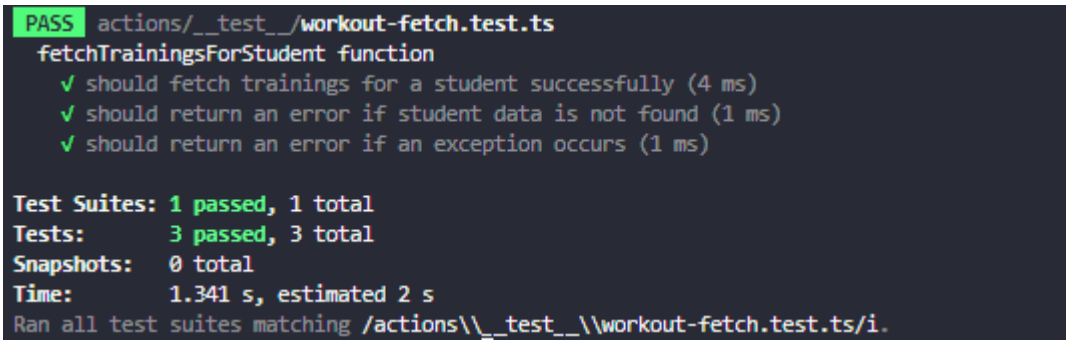
- **userProfile.ts (100% em todas as métricas, exceto 72% para branches):** O arquivo userProfile.ts, onde a função principal reside, foi completamente coberto em termos de declarações, funções e linhas, demonstrando que todos os caminhos principais de execução foram testados. No entanto, a cobertura de branches (condicionais) está em 72%, sugerindo que alguns caminhos alternativos dentro das condicionais não foram testados.

Iteração 2. Resultados



UC08 - Usuário Visualiza Seus Treinos

Os testes unitários foram executados com sucesso para a função `fetchTrainingsForStudent`, como demonstrado na captura de tela. Três casos de teste foram criados para cobrir os principais cenários de uso da função. Abaixo, segue uma análise detalhada dos resultados:



Cenários Testados:

1. Busca bem-sucedida dos treinos para um estudante: O teste verificou que a função retorna corretamente os dados de treino associados a um estudante existente, incluindo detalhes dos blocos de treino, exercícios e máquinas. Isso assegura que a função está integrando corretamente com o banco de dados e retornando os dados esperados.
2. Erro quando os dados do estudante não são encontrados: Este teste validou que a função retorna uma mensagem de erro apropriada quando o estudante não é encontrado ou os dados adicionais do estudante não estão presentes.

Isso é importante para garantir que o sistema forneça feedback adequado quando uma busca falha devido à falta de dados.

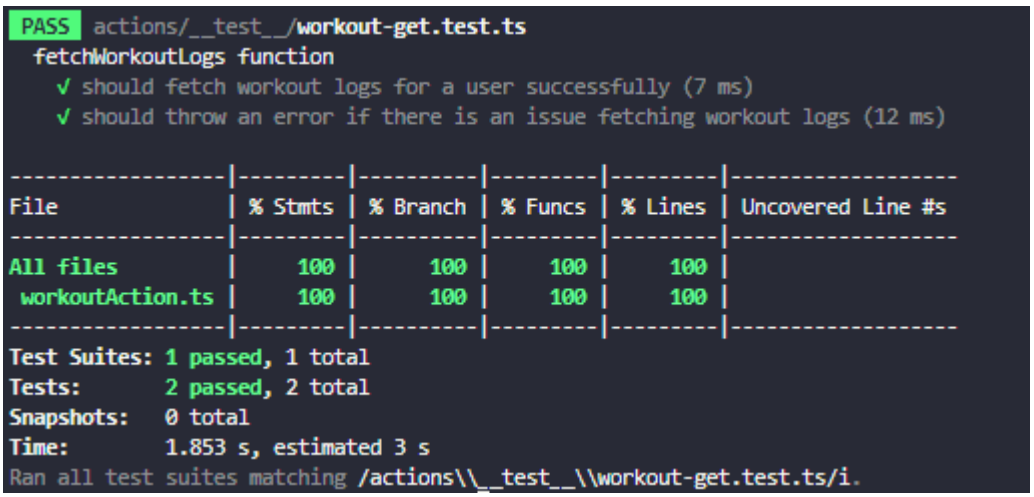
3. Erro ao ocorrer uma exceção durante a busca: Testamos a função para garantir que ela lida corretamente com exceções, como erros de banco de dados, retornando uma mensagem de erro clara e chamando o `console.error` para logar o erro. Isso é crucial para a robustez e a manutenção do código, facilitando a identificação e a correção de problemas.

Cobertura por Arquivo:

- fetch-training.ts (100% em declarações, funções, e linhas, 85.71% em ramificações): A cobertura total da função demonstra que todas as partes críticas do código foram testadas, garantindo confiabilidade e comportamento conforme o esperado. No entanto, há espaço para melhorar a cobertura de ramificações para capturar todos os caminhos possíveis, incluindo diferentes condições e variações de dados.

UC09 - Usuário Visualiza Sua Evolução

Os testes unitários para a função `fetchWorkoutLogs` foram executados com sucesso, como mostrado na captura de tela. Dois casos de teste específicos foram criados para cobrir os principais cenários de uso da função `fetchWorkoutLogs`. Abaixo segue uma análise detalhada dos resultados:



Cenários Testados:

1. **Busca de Logs de Treino do Usuário com Sucesso**:
    - Este teste verifica se a função `fetchWorkoutLogs` é capaz de buscar corretamente os logs de treino associados a um usuário específico. A função

é mockada para retornar um conjunto de dados predefinido, garantindo que os logs sejam recuperados corretamente do banco de dados.

2. **Lançamento de Erro ao Ocorrer um Problema na Busca dos Logs de Treino**:
    - Este teste assegura que a função `fetchWorkoutLogs` lança um erro adequado quando há um problema ao tentar buscar os logs de treino. Para simular um erro, a função mockada `db.workoutLog.findMany` é configurada para lançar uma exceção, o que permite ao teste verificar se a função está gerenciando erros de maneira correta e se o `console.error` é chamado como esperado.

Cobertura por Arquivo:

- **workoutAction.ts (100% em todas as métricas)**: O arquivo `workoutAction.ts`, que contém a função principal, alcançou cobertura total, demonstrando que todos os cenários e ramificações lógicas foram testados adequadamente.

UC10 - Usuário Exclui sua Conta

Os testes unitários para o caso de uso "Usuário Exclui sua Conta" foram executados com sucesso, conforme demonstrado na captura de tela. Dois cenários principais foram testados para garantir o correto funcionamento da função `deleteUserAction`. Abaixo segue uma análise detalhada dos resultados:

Cenários Testados:

1. **Exclusão Bem-Sucedida do Usuário**:
    - **Propósito**: Verificar se a função consegue deletar um usuário corretamente quando fornecido um `userId` válido.

    - **Resultado**: O teste confirmou que o método `db.user.delete` foi chamado com o `userId` correto e que a função retornou `{ success: true }` ao concluir a exclusão com sucesso.
2. **Erro Durante a Exclusão**:
    - **Propósito**: Garantir que a função lida corretamente com erros durante o processo de exclusão, como falhas no banco de dados.
    - **Resultado**: O teste simulou um erro ao tentar deletar o usuário e verificou que a função chamou `console.error` com a mensagem de erro apropriada. Além disso, a função retornou um objeto de erro `{ error: "Erro ao excluir usuário." }`, garantindo que falhas são tratadas adequadamente.

Cobertura por Arquivo:

- **delete_user.ts (100% em todas as métricas)**: O arquivo que contém a função `deleteUserAction` teve 100% de cobertura em todas as métricas (declarações, funções, linhas, e branches). Isso indica que todas as partes relevantes da função foram testadas, validando tanto o caminho de sucesso quanto o caminho de erro.

Iteração 3.

```
 PASS  actions/__test__/workoutLogs-get.test.ts
 PASS  actions/__test__/workout-fetch.test.ts
 PASS  actions/__test__/delete-user.test.ts
------------------|---------|----------|---------|---------|-------------------
File              | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
------------------|---------|----------|---------|---------|-------------------
All files         |     100 |    85.71 |     100 |     100 |
 delete_user.ts   |     100 |      100 |     100 |     100 |
 fetch-traning.ts |     100 |    85.71 |     100 |     100 | 7
 workoutAction.ts |     100 |      100 |     100 |     100 |
------------------|---------|----------|---------|---------|-------------------

Test Suites: 3 passed, 3 total
Tests:       7 passed, 7 total
Snapshots:   0 total
Time:        1.614 s
Ran all test suites matching /actions\\__test__\\workout-fetch.test.ts|actions\\__test_
s/i.
```

ALL runs:

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	88.75	70.17	52.17	88.67	
web2-2un	100	100	100	100	
routes.ts	100	100	100	100	
web2-2un/actions	97.5	85.1	100	97.47	
delete_user.ts	100	100	100	100	
fetch-traning.ts	100	85.71	100	100	7
login.ts	91.42	89.65	100	91.42	74,102,108
logout.ts	100	100	100	100	
new-password.ts	100	100	100	100	
register.ts	100	100	100	100	
tokenTwoFactor.ts	100	100	100	100	
updateProfile.ts	100	82.35	100	100	40-42
userProfile.ts	100	72	100	100	32-37,42
workoutAction.ts	100	100	100	100	
web2-2un/enums	0	0	0	0	
role.ts	0	0	0	0	
web2-2un/schemas	58.33	0	15.38	58.33	
index.ts	58.33	0	15.38	58.33	16-23,47-55,115-117,135-143,154,165

Test Suites: 1 failed, 9 passed, 10 total
Tests: 1 failed, 38 passed, 39 total
Snapshots: 0 total