

DOCUMENTAÇÃO DE TESTES DO PROJETO INTEGRADOR.

NeonPay Academy



Grupo 1:

Alexandra Christine – 24026156

Hebert Esteves - 24026079

Gabrielly Cintra - 24025696

José Bento – 24026127

Sumário

<u>Sumário</u>	<u>1</u>
<u>1.Introdução</u>	<u>1</u>
<u>2.Objetivo</u>	<u>2</u>
<u>3.Teste Unitário - Cadastro</u>	<u>2</u>
<u>3.1. Entrada</u>	<u>3</u>
<u>3.2. Saída:</u>	<u>3</u>
<u>3.3. Descrição</u>	<u>3</u>
<u>3.4. Cenários que foram testados</u>	<u>3</u>
<u>3.5. Como o teste é realizado</u>	<u>4</u>
<u>4.0. Teste Unitário - Login e Senha</u>	<u>5</u>
<u>4.1. Entrada</u>	<u>6</u>
<u>4.2. Saída:</u>	<u>6</u>
<u>4.3. Descrição</u>	<u>6</u>
<u>4.4 .Cenários que foram testados</u>	<u>7</u>
<u>4.5. Como o teste é realizado</u>	<u>7</u>
<u>5.0. Teste Unitário - Consulta de Pontos</u>	<u>7</u>
<u>5.1. Entrada</u>	<u>8</u>
<u>5.2. Saída:</u>	<u>8</u>
<u>5.3. Descrição</u>	<u>8</u>
<u>5.4 .Cenários que foram testados</u>	<u>9</u>
<u>5.5. Como o teste é realizado</u>	<u>9</u>
<u>6.0. Teste Unitário - Troca de Pontos</u>	<u>9</u>
<u>6.1. Entrada</u>	<u>10</u>
<u>6.2. Saída:</u>	<u>10</u>
<u>6.3. Descrição</u>	<u>10</u>
<u>6.4. Cenários que foram testados</u>	<u>11</u>
<u>6.5. Como o teste é realizado</u>	<u>11</u>
<u>7.0. Teste de integração - Transação Financeira Pix</u>	<u>11</u>
<u>7.1. Entrada</u>	<u>12</u>
<u>7.2. Saída:</u>	<u>12</u>
<u>7.3. Descrição</u>	<u>12</u>

<u>7.4. Cenários que foram testados</u>	<u>13</u>
<u>7.5 Visualização gráfica no banco de dados - Usuários</u>	<u>14</u>
<u>7.6 Visualização gráfica no banco de dados - Transações</u>	<u>14</u>
<u>8.0. Teste de integração - Senha incorreta Pix.</u>	<u>17</u>
<u>8.1. Entrada</u>	<u>18</u>
<u>8.2. Saída</u>	<u>18</u>
<u>8.3. Descrição</u>	<u>18</u>
<u>8.4. Cenário que foi testado</u>	<u>19</u>
<u>8.5. Como o teste é realizado</u>	<u>19</u>
<u>8.6 Visualização gráfica no banco de dados - Usuários</u>	<u>20</u>
<u>8.7 Visualização gráfica no banco de dados - Transações</u>	<u>20</u>
<u>9.0. Teste de Carga – Teste de Integração PIX.</u>	<u>21</u>
<u>9.1. Entrada</u>	<u>22</u>
<u>9.2. Saída</u>	<u>23</u>
<u>9.3 Descrição</u>	<u>24</u>
<u>9.4. Cenário que foi testado</u>	<u>25</u>
<u>9.5. Como o teste é realizado</u>	<u>26</u>
<u>9.6. Saída no Terminal</u>	<u>26</u>
<u>10.0. Teste de Carga – Teste Unitário PIX.</u>	<u>27</u>
<u>10.1. Entrada</u>	<u>27</u>
<u>10.2. Saída</u>	<u>28</u>
<u>10.3. Descrição</u>	<u>29</u>
<u>10.4. Cenários que foram testados</u>	<u>29</u>
<u>10.5. Como o teste é realizado</u>	<u>30</u>
<u>10.6 Funcionalidades testadas</u>	<u>30</u>
<u>10.7. Saída no Terminal</u>	<u>31</u>

1.0 Introdução

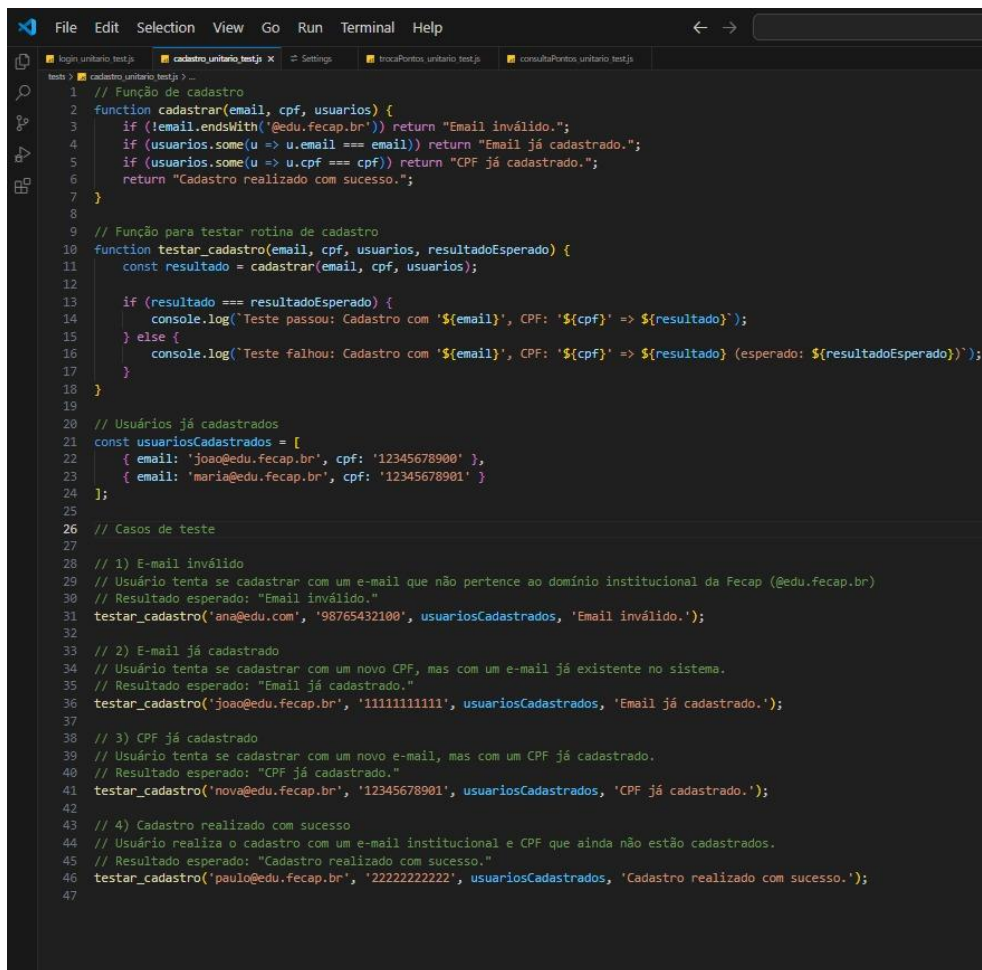
A NeonPay Academy é um aplicativo mobile desenvolvido para estudantes da FECAP, com o objetivo de facilitar pagamentos e incentivar o engajamento acadêmico por meio de um sistema de recompensas. O app oferece uma solução moderna e segura para transações financeiras no ambiente universitário. Entre suas funcionalidades, o NeonPay permite que os usuários realizem pagamentos via Pix diretamente pelo aplicativo, acumulem pontos conforme os valores gastos e resgatem recompensas utilizando esses pontos, como produtos da atlética, brindes ou benefícios exclusivos. O design do aplicativo foi pensado para o público jovem, com uma interface moderna, intuitiva e fácil de navegar. O backend é desenvolvido em Node.js com banco de dados MySQL, enquanto o frontend é um app Android nativo em Java, utilizando bibliotecas como MPAndroidChart e Volley para a visualização de dados e requisições à API.

2.0 Objetivo.

O objetivo deste documento é apresentar os testes realizados no sistema, com foco nos testes de integração e testes unitários. Os testes de integração demonstram como os diferentes componentes do app como autenticação, transações, sistema de pontuação funcionam em conjunto de forma correta, garantindo que o fluxo completo do usuário ocorra sem falhas. Já os testes unitários têm como finalidade validar individualmente funções e métodos isolados do backend, como verificações de autenticação, geração de tokens, cálculos de pontuação e validação de entradas, assegurando que cada parte do sistema se comporte conforme o esperado mesmo antes da integração total. Ao reunir ambos os tipos de testes, garantimos uma base sólida para a qualidade, a confiabilidade e a escalabilidade do nosso aplicativo.

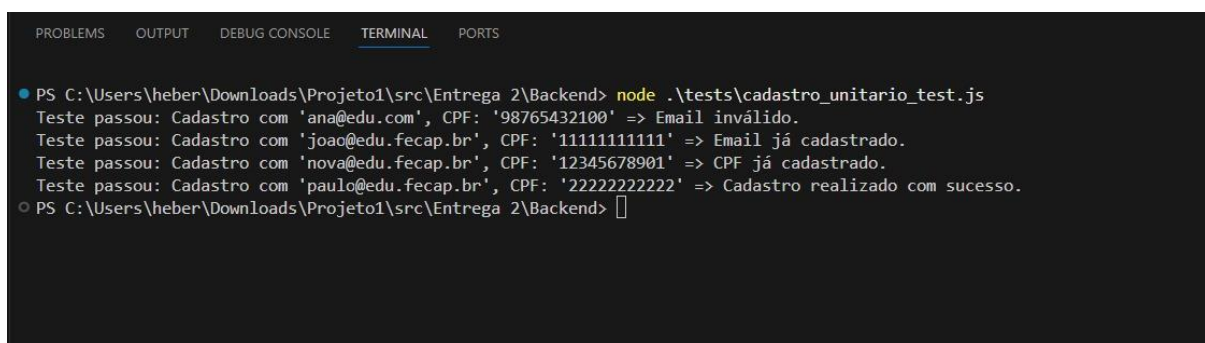
3.0 Teste Unitário - Cadastro.

3.1. Entrada



```
1 // Função de cadastro
2 function cadastrar(email, cpf, usuarios) {
3   if (!email.endsWith('@edu.fecap.br')) return "Email inválido.";
4   if (usuarios.some(u => u.email === email)) return "Email já cadastrado.";
5   if (usuarios.some(u => u.cpf === cpf)) return "CPF já cadastrado.";
6   return "Cadastro realizado com sucesso.";
7 }
8
9 // Função para testar rotina de cadastro
10 function testar_cadastro(email, cpf, usuarios, resultadoEsperado) {
11   const resultado = cadastrar(email, cpf, usuarios);
12
13   if (resultado === resultadoEsperado) {
14     console.log(`Teste passou: Cadastro com '${email}', CPF: '${cpf}' => ${resultado}`);
15   } else {
16     console.log(`Teste falhou: Cadastro com '${email}', CPF: '${cpf}' => ${resultado} (esperado: ${resultadoEsperado})`);
17   }
18 }
19
20 // Usuários já cadastrados
21 const usuariosCadastrados = [
22   { email: 'joao@edu.fecap.br', cpf: '12345678900' },
23   { email: 'maria@edu.fecap.br', cpf: '12345678901' }
24 ];
25
26 // Casos de teste
27
28 // 1) E-mail inválido
29 // Usuário tenta se cadastrar com um e-mail que não pertence ao domínio institucional da Fecap (@edu.fecap.br)
30 // Resultado esperado: "Email inválido."
31 testar_cadastro('ana@edu.com', '98765432100', usuariosCadastrados, 'Email inválido.');
```

3.2. Saída:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
● PS C:\Users\heber\Downloads\Projeto1\src\Entrega 2\Backend> node .\tests\cadastro_unitario_test.js
  Teste passou: Cadastro com 'ana@edu.com', CPF: '98765432100' => Email inválido.
  Teste passou: Cadastro com 'joao@edu.fecap.br', CPF: '11111111111' => Email já cadastrado.
  Teste passou: Cadastro com 'nova@edu.fecap.br', CPF: '12345678901' => CPF já cadastrado.
  Teste passou: Cadastro com 'paulo@edu.fecap.br', CPF: '22222222222' => Cadastro realizado com sucesso.
○ PS C:\Users\heber\Downloads\Projeto1\src\Entrega 2\Backend> 
```

3.3. Descrição

Este é um teste unitário escrito em JavaScript para verificar o comportamento da função de **cadastro** de usuários em um sistema. Ele simula diferentes cenários para garantir que o cadastro funcione corretamente, validando o domínio do e-mail, a unicidade do CPF e do e-mail.

3.4. Cenários que foram testados

1. E-mail inválido

Usuário tenta se cadastrar com um e-mail que não pertence ao domínio institucional da FECAP (@edu.fecap.br).

Resultado esperado: "Email inválido."

2. E-mail já cadastrado

Usuário tenta se cadastrar com um novo CPF, mas com um e-mail já existente no sistema.

Resultado esperado: "Email já cadastrado."

3. CPF já cadastrado

Usuário tenta se cadastrar com um novo e-mail, mas com um CPF já existente.

Resultado esperado: "CPF já cadastrado."

4. Cadastro realizado com sucesso

Usuário realiza o cadastro com um e-mail institucional e CPF que ainda não estão cadastrados.

Resultado esperado: "Cadastro realizado com sucesso."

O terminal mostra os testes com console.log, indicando se o cadastro foi realizado corretamente ou se os erros esperados ocorreram. Como todos os testes passaram, o terminal exibe:

Teste passou: Cadastro com 'ana@edu.com', CPF: '98765432100' => Email inválido.

Teste passou: Cadastro com 'joao@edu.fecap.br', CPF: '11111111111' => Email já cadastrado.

Teste passou: Cadastro com 'nova@edu.fecap.br', CPF: '12345678901' => CPF já cadastrado.

Teste passou: Cadastro com 'paulo@edu.fecap.br', CPF: '22222222222' => Cadastro realizado com sucesso.

3.5. Como o teste é realizado

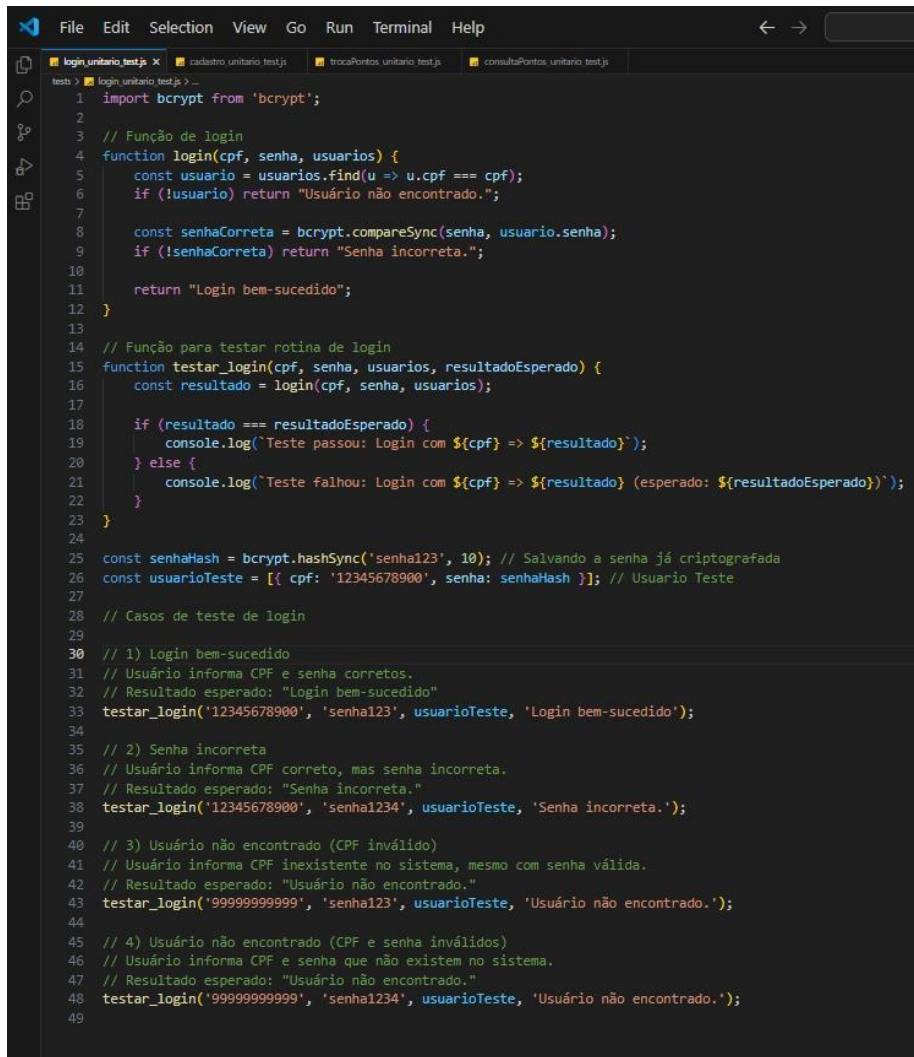
A função `testar_cadastro()` recebe os parâmetros do teste (email, cpf, a lista de usuários cadastrados e o resultadoEsperado). Ela executa a função `cadastar()` e compara o retorno com o resultado esperado. Dependendo do caso, imprime no terminal se o teste passou ou falhou.

- Se o resultado for o esperado, exibe:
Teste passou: Cadastro com '<email>', CPF: '<cpf>' => <mensagem>

- Se o resultado for diferente do esperado, exibe:
Teste falhou: Cadastro com '<email>', CPF: '<cpf>' => <mensagem> (esperado:
<resultadoEsperado>)

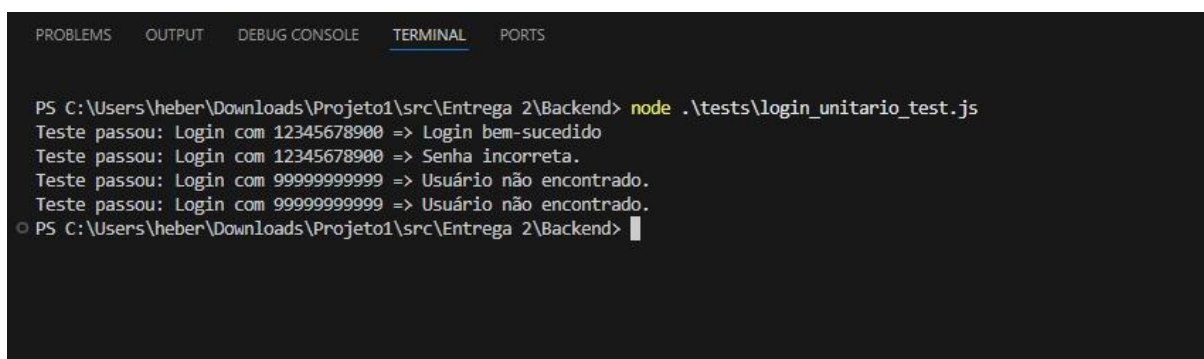
4.0. Teste Unitário - Login e Senha.

4.1. Entrada



```
1 import bcrypt from 'bcrypt';
2
3 // Função de login
4 function login(cpf, senha, usuarios) {
5   const usuario = usuarios.find(u => u.cpf === cpf);
6   if (!usuario) return "Usuário não encontrado.";
7
8   const senhaCorreta = bcrypt.compareSync(senha, usuario.senha);
9   if (!senhaCorreta) return "Senha incorreta.";
10
11   return "Login bem-sucedido";
12 }
13
14 // Função para testar rotina de login
15 function testar_login(cpf, senha, usuarios, resultadoEsperado) {
16   const resultado = login(cpf, senha, usuarios);
17
18   if (resultado === resultadoEsperado) {
19     console.log(`Teste passou: Login com ${cpf} => ${resultado}`);
20   } else {
21     console.log(`Teste falhou: Login com ${cpf} => ${resultado} (esperado: ${resultadoEsperado})`);
22   }
23 }
24
25 const senhaHash = bcrypt.hashSync('senha123', 10); // Salvando a senha já criptografada
26 const usuarioTeste = [{ cpf: '12345678900', senha: senhaHash }]; // Usuario Teste
27
28 // Casos de teste de login
29
30 // 1) Login bem-sucedido
31 // Usuário informa CPF e senha corretos.
32 // Resultado esperado: "Login bem-sucedido"
33 testar_login('12345678900', 'senha123', usuarioTeste, 'Login bem-sucedido');
34
35 // 2) Senha incorreta
36 // Usuário informa CPF correto, mas senha incorreta.
37 // Resultado esperado: "Senha incorreta."
38 testar_login('12345678900', 'senha1234', usuarioTeste, 'Senha incorreta.');
```

4.2. Saída:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\heber\Downloads\Projeto1\src\Entrega 2\Backend> node .\tests\login_unitario_test.js
Teste passou: Login com 12345678900 => Login bem-sucedido
Teste passou: Login com 12345678900 => Senha incorreta.
Teste passou: Login com 99999999999 => Usuário não encontrado.
Teste passou: Login com 99999999999 => Usuário não encontrado.
PS C:\Users\heber\Downloads\Projeto1\src\Entrega 2\Backend>
```


4.3. Descrição

Este é um teste unitário escrito em JavaScript para verificar o comportamento da função de login de um sistema. Ele simula diferentes cenários para garantir que o login funcione corretamente em cada um deles.

4.4 . Cenários que foram testados

1. Login bem-sucedido CPF e senha corretos.

Resultado: "Login bem-sucedido".

2. Senha incorreta CPF válido, senha errada.

Resultado: "Senha incorreta".

3. Usuário não encontrado (CPF inválido) CPF não cadastrado, mesmo com senha válida.

Resultado: "Usuário não encontrado".

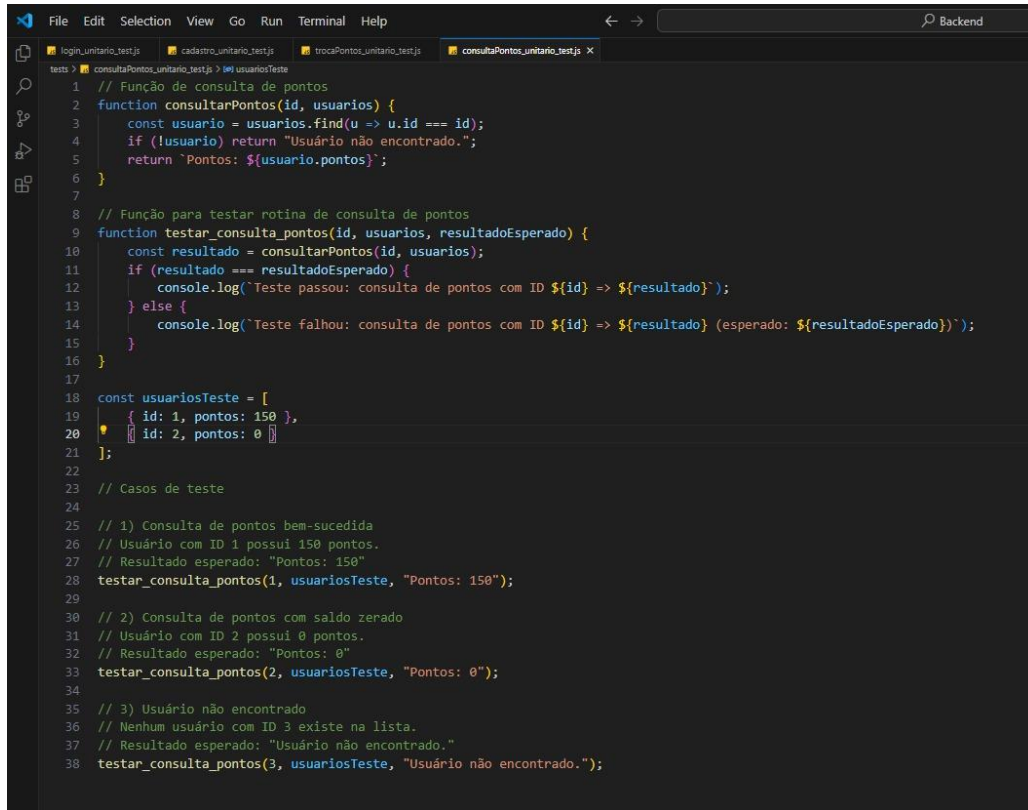
4. Usuário não encontrado (CPF e senha inválidos) CPF e senha inexistentes. Resultado: "Usuário não encontrado".

4.5. Como o teste é realizado

A função `testar_login()` recebe os parâmetros do teste (`cpf`, `senha`, `base de usuários` e `resultado esperado`), executa a função `login()` e compara o retorno com o resultado esperado. Dependendo do caso, imprime no terminal se o teste passou ou falhou.

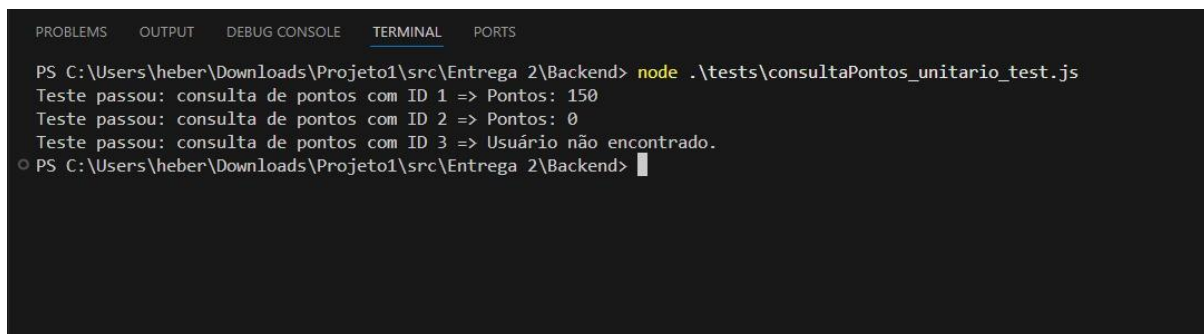
5.0. Teste Unitário - Consulta de Pontos.

5.1. Entrada



```
1 // Função de consulta de pontos
2 function consultarPontos(id, usuarios) {
3   const usuario = usuarios.find(u => u.id === id);
4   if (!usuario) return "Usuário não encontrado.";
5   return `Pontos: ${usuario.pontos}`;
6 }
7
8 // Função para testar rotina de consulta de pontos
9 function testar_consulta_pontos(id, usuarios, resultadoEsperado) {
10  const resultado = consultarPontos(id, usuarios);
11  if (resultado === resultadoEsperado) {
12    console.log(`Teste passou: consulta de pontos com ID ${id} => ${resultado}`);
13  } else {
14    console.log(`Teste falhou: consulta de pontos com ID ${id} => ${resultado} (esperado: ${resultadoEsperado})`);
15  }
16 }
17
18 const usuariosTeste = [
19   { id: 1, pontos: 150 },
20   { id: 2, pontos: 0 },
21 ];
22
23 // Casos de teste
24
25 // 1) Consulta de pontos bem-sucedida
26 // Usuário com ID 1 possui 150 pontos.
27 // Resultado esperado: "Pontos: 150"
28 testar_consulta_pontos(1, usuariosTeste, "Pontos: 150");
29
30 // 2) Consulta de pontos com saldo zerado
31 // Usuário com ID 2 possui 0 pontos.
32 // Resultado esperado: "Pontos: 0"
33 testar_consulta_pontos(2, usuariosTeste, "Pontos: 0");
34
35 // 3) Usuário não encontrado
36 // Nenhum usuário com ID 3 existe na lista.
37 // Resultado esperado: "Usuário não encontrado."
38 testar_consulta_pontos(3, usuariosTeste, "Usuário não encontrado.");
```

5.2. Saída:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS C:\Users\heber\Downloads\Projeto1\src\Entrega 2\Backend> node .\tests\consultaPontos_unitario_test.js
Teste passou: consulta de pontos com ID 1 => Pontos: 150
Teste passou: consulta de pontos com ID 2 => Pontos: 0
Teste passou: consulta de pontos com ID 3 => Usuário não encontrado.
PS C:\Users\heber\Downloads\Projeto1\src\Entrega 2\Backend>
```

5.3. Descrição

Este é um **teste unitário** escrito em JavaScript para verificar o comportamento da função de **consulta de pontos** de um sistema. Ele simula diferentes cenários para garantir que o sistema retorne corretamente o número de pontos do usuário ou informe quando ele não for encontrado.

5.4 . Cenários que foram testados

1. **Consulta de pontos bem-sucedida**
O usuário com ID 1 possui 150 pontos.
Resultado esperado: "Pontos: 150"
2. **Consulta de pontos com saldo zerado**
O usuário com ID 2 possui 0 pontos.
Resultado esperado: "Pontos: 0"
3. **Usuário não encontrado**
Nenhum usuário com ID 3 existe na lista.
Resultado esperado: "Usuário não encontrado."

O terminal mostra os testes com `console.log`, indicando se a consulta foi executada corretamente em cada caso. Como todos os testes passaram, o terminal exibiu:

Teste passou: consulta de pontos com ID 1 => Pontos: 150

Teste passou: consulta de pontos com ID 2 => Pontos: 0

Teste passou: consulta de pontos com ID 3 => Usuário não encontrado.

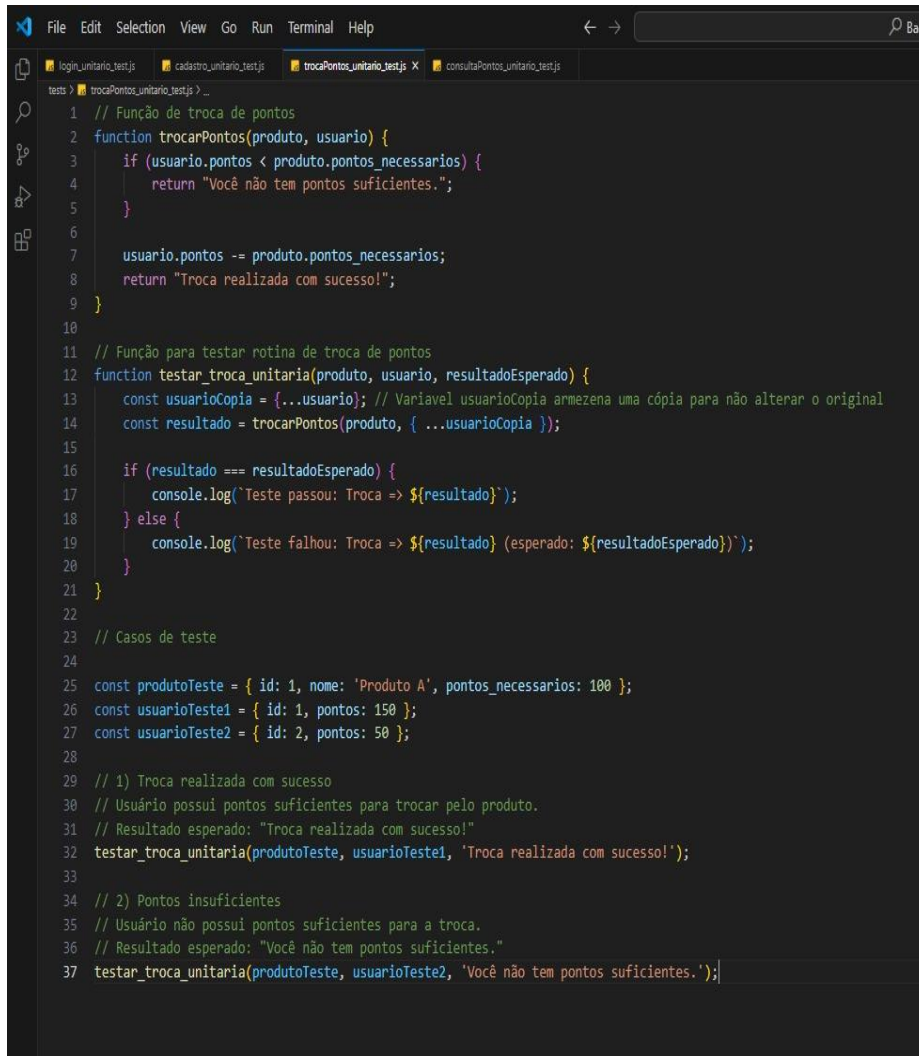
5.5. Como o teste é realizado

A função **testar_consulta_pontos()** recebe os parâmetros do teste (id, usuários e resultado esperado), executa a função **consultarPontos()** e compara o retorno com o resultado esperado.

Com base na comparação, imprime no terminal se o teste **passou** ou **falhou**. Isso garante que tanto as consultas positivas quanto as negativas sejam corretamente avaliadas.

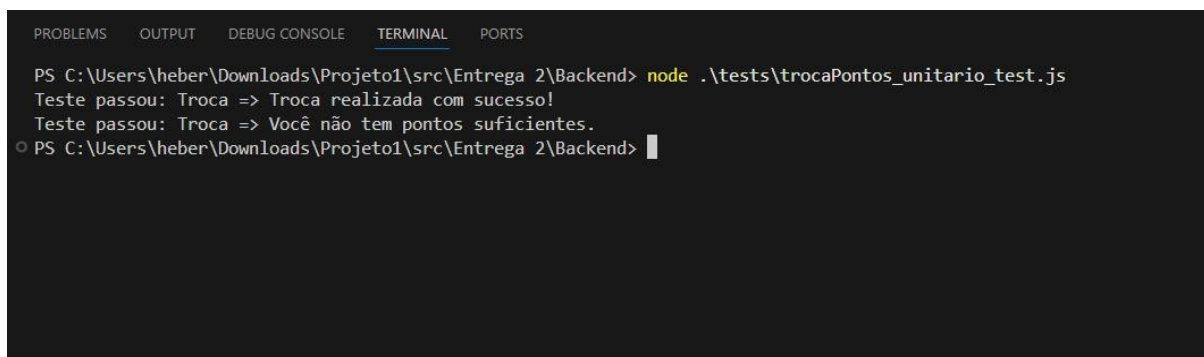
6.0. Teste Unitário - Troca de Pontos.

6.1. Entrada



```
1 // Função de troca de pontos
2 function trocarPontos(produto, usuario) {
3   if (usuario.pontos < produto.pontos_necessarios) {
4     return "Você não tem pontos suficientes.";
5   }
6
7   usuario.pontos -= produto.pontos_necessarios;
8   return "Troca realizada com sucesso!";
9 }
10
11 // Função para testar rotina de troca de pontos
12 function testar_troca_unitaria(produto, usuario, resultadoEsperado) {
13   const usuarioCopia = {...usuario}; // Variavel usuarioCopia armazena uma cópia para não alterar o original
14   const resultado = trocarPontos(produto, { ...usuarioCopia });
15
16   if (resultado === resultadoEsperado) {
17     console.log("Teste passou: Troca => ${resultado}");
18   } else {
19     console.log("Teste falhou: Troca => ${resultado} (esperado: ${resultadoEsperado})");
20   }
21 }
22
23 // Casos de teste
24
25 const produtoTeste = { id: 1, nome: 'Produto A', pontos_necessarios: 100 };
26 const usuarioTeste1 = { id: 1, pontos: 150 };
27 const usuarioTeste2 = { id: 2, pontos: 50 };
28
29 // 1) Troca realizada com sucesso
30 // Usuário possui pontos suficientes para trocar pelo produto.
31 // Resultado esperado: "Troca realizada com sucesso!"
32 testar_troca_unitaria(produtoTeste, usuarioTeste1, 'Troca realizada com sucesso!');
33
34 // 2) Pontos insuficientes
35 // Usuário não possui pontos suficientes para a troca.
36 // Resultado esperado: "Você não tem pontos suficientes."
37 testar_troca_unitaria(produtoTeste, usuarioTeste2, 'Você não tem pontos suficientes.');
```

6.2. Saída:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS C:\Users\heber\Downloads\Projeto1\src\Entrega 2\Backend> node .\tests\trocaPontos_unitario_test.js
Teste passou: Troca => Troca realizada com sucesso!
Teste passou: Troca => Você não tem pontos suficientes.
PS C:\Users\heber\Downloads\Projeto1\src\Entrega 2\Backend>
```

6.3. Descrição

Este é um teste unitário escrito em JavaScript para verificar o comportamento da função de troca de pontos de um sistema. Ele simula diferentes cenários para garantir que o sistema permita ou negue a troca de pontos de acordo com a quantidade de pontos disponíveis do usuário.

6.4. Cenários que foram testados

1. **Troca realizada com sucesso O usuário possui pontos suficientes para realizar a troca pelo produto.**

Resultado: "Troca realizada com sucesso!"

2. **Pontos insuficientes O usuário tenta realizar a troca, mas não possui pontos suficientes.**

Resultado: "Você não tem pontos suficientes."

O terminal mostra os testes com `console.log`, indicando se a troca foi realizada com sucesso ou corretamente negada. Como todos os testes passaram, o terminal exibe:

javascript Copiar Editar Teste passou: Troca => Troca realizada com sucesso!

Teste passou: Troca => Você não tem pontos suficientes.

6.5. Como o teste é realizado

A função **testar_troca_unitaria()** recebe os parâmetros do teste (produto, usuário e resultado esperado), realiza uma cópia do objeto do usuário para evitar modificações diretas no original, executa a função **trocarPontos()** e compara o retorno com o resultado esperado. O sistema imprime no terminal se o teste passou ou falhou, com base na comparação entre o retorno e o valor esperado.

7.0. Teste de integração - Transação Financeira Pix.

7.1. Entrada

```
File Edit Selection View Go Run Terminal Help
pix.integration.test.js x
describe('Testes de integração - PIX', () => {
  // Importação do supertest, fazer requisições HTTP simuladas à nossa API
  import request from 'supertest'; // Importação do app que representa nosso servidor
  import app from '../src/Entrega 2/Backend/server.js'; // Importação da conexão com o banco de dados
  import db from '../src/Entrega 2/Backend/config/db.js'; // Importação do bcrypt para criptografar a senha (como é feito no sistema real)
  import bcrypt from 'bcrypt';

  // Início da suíte de testes com Jest
  describe('Testes de integração - PIX', () => {
    // Variáveis que vamos usar entre os testes
    let usuarioId;
    let transacaoId;

    // Definindo a senha e o valor que será usado nos testes
    const senha = 'senha123';
    const valorPix = 50;

    // Função que roda antes de todos os testes
    beforeAll(async () => {
      // Criptografa a senha como se fosse no cadastro real
      const senhaCriptografada = await bcrypt.hash(senha, 10);

      // Insere um usuário de teste no banco de dados
      const [resultado] = await db.execute(
        "INSERT INTO usuarios (nome, cpf, email, telefone, senha, saldo, pontos) VALUES (?, ?, ?, ?, ?, ?, ?)",
        ['Teste PIX', '00000000000', 'pix@test.com', '11999999999', senhaCriptografada, 100, 0]
      );

      // Guarda o ID do usuário criado para ser usado nos testes
      usuarioId = resultado.insertId;
    });

    // Primeiro teste: criar uma cobrança Pix
    it('Deve gerar uma cobrança Pix com sucesso', async () => {
      // É enviado uma requisição POST para gerar a cobrança
      const resultadoCobranca = await request(app)
        .post('/pix/gerar-cobranca')
        .send({ usuario_id: usuarioId, valor: valorPix });

      expect(resultadoCobranca.status).toBe(200); // Espera-se que a resposta tenha status 200 (sucesso)

      // Verifica se o QR Code, a chave Pix e o ID da transação foram retornados
      expect(resultadoCobranca.body.qr_code).toBeDefined();
      expect(resultadoCobranca.body.chave_pix).toBeDefined();
      expect(resultadoCobranca.body.transacao_id).toBeDefined();

      // Guarda o ID da transação para ser usado no próximo teste
      transacaoId = resultadoCobranca.body.transacao_id;
    });

    // Segundo teste: simular o recebimento do pagamento via webhook
    it('Deve confirmar o pagamento via webhook e atualizar saldo', async () => {
      // Envia uma requisição POST para simular o webhook de confirmação
      const resultadoWebhook = await request(app)
        .post('/pix/webhook')
        .send({ transacao_id: transacaoId });

      expect(resultadoWebhook.status).toBe(200); // Espera-se que a resposta tenha status 200

      expect(resultadoWebhook.body.mensagem).toBe("Transação confirmada."); // Verifica se a mensagem de confirmação foi recebida

      const saldoUsuario = await request(app).get('/pix/saldo/${usuarioId}'); // Consulta o saldo do usuário após o pagamento
    });
  });
});
```

7.2. Saída

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PASS ./pix.integration.test.js
Testes de integração - PIX
  ✓ Deve gerar uma cobrança Pix com sucesso (24 ms)
  ✓ Deve confirmar o pagamento via webhook e atualizar saldo (10 ms)
  ✓ Deve enviar um Pix e aumentar os pontos do usuário (71 ms)

Test Suites: 1 passed, 1 total
Tests: 3 passed, 3 total
Snapshots: 0 total
Time: 0.953 s, estimated 1 s
Ran all test suites.
Jest did not exit one second after the test run has completed.
```

7.3. Descrição

Este é um **teste de integração** desenvolvido com **Jest** e **Supertest** para validar o funcionamento completo da funcionalidade de **PIX** no sistema NeonPay Academy. Ele cobre três fluxos principais:

1. Geração de uma cobrança via Pix.
2. Simulação de confirmação de pagamento por webhook.
3. Envio de um Pix para outro usuário, com impacto em saldo e pontos.

O teste simula interações reais com a API, incluindo escrita e leitura no banco de dados, além de verificar as alterações no saldo e pontuação dos usuários.

7.4. Cenários que foram testados

1. Geração de cobrança Pix com sucesso

Uma requisição POST `/pix/gerar-cobranca` é enviada com o `usuario_id` e valor da cobrança.

A API retorna status 200, juntamente com o QR Code e ID da transação.

O ID da transação é armazenado para uso em etapas seguintes.

Resultado esperado:

- status: 200
- Retorno com `qr_code`, `chave_pix` e `transacao_id`.

2. Confirmação de pagamento via webhook

- Uma requisição POST `/pix/webhook` é enviada com o `transacao_id`.
- A API atualiza o status da transação e adiciona o valor ao saldo do usuário.
- Uma verificação posterior é feita com GET `/pix/saldo/:id`.

Resultado esperado:

- status: 200
- Mensagem: "Transação confirmada."
- Novo saldo: 150 (100 original + 50 da transação) **Envio de Pix para outro usuário**

3. Uma requisição POST /pix/enviar é feita com o usuario_id, valor, chave de destino e senha.

- O sistema realiza a transação, reduz o saldo do usuário e converte o valor em pontos (1 ponto a cada R\$ 5). Em seguida, as rotas GET /pix/pontos/:id e GET /pix/saldo/:id são usadas para conferir os valores atualizados.

Resultado esperado:

- status: 200
- Mensagem: "Pix enviado."
- Novo saldo: 125 (150 - 25)
- Pontos: 5 (25 / 5)

7.5 Visualização gráfica no banco de dados - Usuários

Mostrando registros 0 - 1 (2 no total, Consulta levou 0 0002 segundos)

SELECT * FROM `usuarios`

Perfil [Editar em linha] [Editar] [Demonstrar SQL] [Criar código PHP] [Atualizar]

☐ Mostrar tudo | Número de linhas: 25 | Filtrar linhas: Procurar nesta tabela | Ordenar pela chave: Nenhum

Opções extras

	id	nome	cpf	data_nasc	email	telefone	senha	saldo	pontos
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Remover	1	Hebert Esteves	47877529897	2006-01-08	hebert@email.com	11987654321	\$2b\$10\$KZgWGJpU9RFzvuMaly3buDv9VbTOTXIO0lolumVE8	800.00	199570
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Remover	23	Teste PIX	00000000000	0000-00-00	pix@test.com	11999999999	\$2b\$10\$1NbaJc83CB09pDaT2x2sWe8zcqMvU1ZhyrSFKlabLY...	125.00	5

☐ Marcar todos | Com marcados: ☐ Editar ☐ Copiar ☐ Remover ☐ Exportar

☐ Mostrar tudo | Número de linhas: 25 | Filtrar linhas: Procurar nesta tabela | Ordenar pela chave: Nenhum

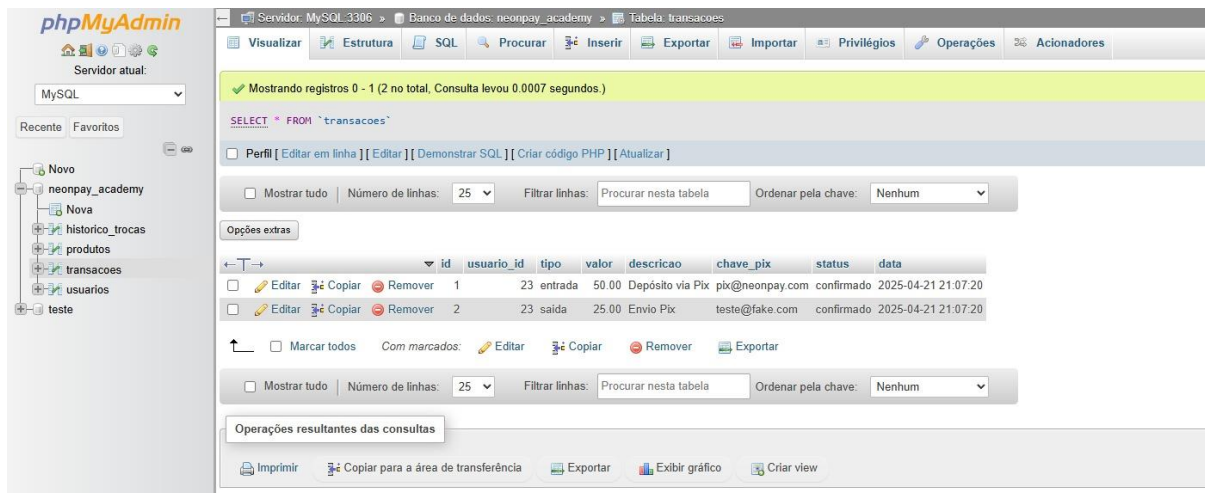
Operações resultantes das consultas

☐ Imprimir ☐ Copiar para a área de transferência ☐ Exportar ☐ Exibir gráfico ☐ Criar view

Pelo phpMyAdmin, observa-se o seguinte:

- O usuário "Teste PIX" possui saldo atualizado (125.00) e pontos (5) após os testes.
- As informações de e-mail, CPF, senha e telefone foram cadastradas conforme o script de inserção do teste.

7.6 Visualização gráfica no banco de dados - Transações



The screenshot shows the phpMyAdmin interface for a MySQL database named 'neonpay_academy'. The 'transacoes' table is selected, and the SQL query 'SELECT * FROM transacoes' is executed. The results show two records:

	id	usuario_id	tipo	valor	descricao	chave_pix	status	data
<input type="checkbox"/>	1	23	entrada	50.00	Depósito via Pix	pix@neonpay.com	confirmado	2025-04-21 21:07:20
<input type="checkbox"/>	2	23	saida	25.00	Envio Pix	teste@fake.com	confirmado	2025-04-21 21:07:20

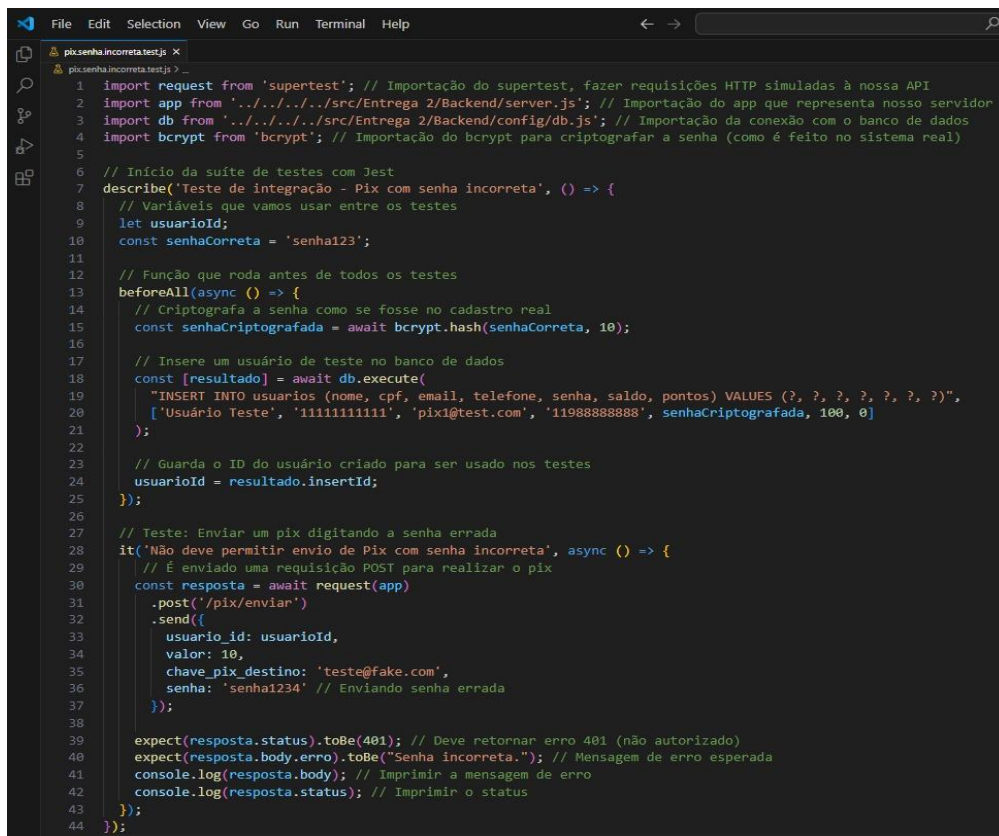
Através da interface do **phpMyAdmin**, observamos que duas transações foram corretamente registradas na tabela transações:

- **Depósito via Pix**
- **ID:** 1
- **Usuário ID:** 23
- **Tipo:** entrada
- **Valor:** R\$ 50,00
- **Descrição:** Depósito via Pix
- **Chave Pix:** pix@neonpay.com
- **Status:** confirmado
- **Data:** 2025-04-21 21:07:20
- **Envio de Pix**
- **ID:** 2
- **Usuário ID:** 23
- **Tipo:** saída
- **Valor:** R\$ 25,00
- **Descrição:** Envio Pix

- **Chave Pix:** [teste@fake.com](mailto:test@fake.com)
- **Status:** confirmado
- **Data:** 2025-04-21 21:07:20

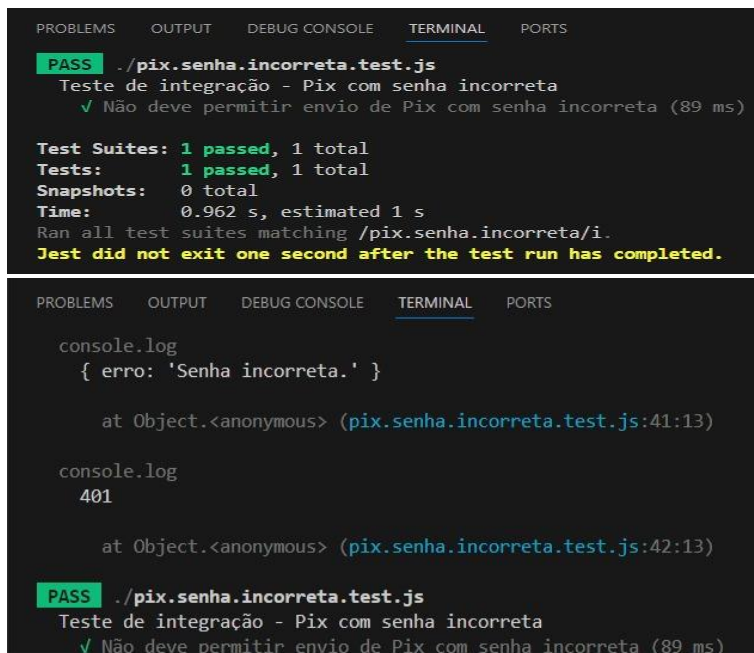
8.0. Teste de integração - Senha incorreta Pix.

8.1. Entrada



```
1 import request from 'supertest'; // Importação do supertest, fazer requisições HTTP simuladas à nossa API
2 import app from '../../src/Entrega 2/Backend/server.js'; // Importação do app que representa nosso servidor
3 import db from '../../src/Entrega 2/Backend/config/db.js'; // Importação da conexão com o banco de dados
4 import bcrypt from 'bcrypt'; // Importação do bcrypt para criptografar a senha (como é feito no sistema real)
5
6 // Início da suíte de testes com Jest
7 describe('Teste de integração - Pix com senha incorreta', () => {
8   // Variáveis que vamos usar entre os testes
9   let usuarioId;
10   const senhaCorreta = 'senha123';
11
12   // Função que roda antes de todos os testes
13   beforeEach(async () => {
14     // Criptografa a senha como se fosse no cadastro real
15     const senhaCriptografada = await bcrypt.hash(senhaCorreta, 10);
16
17     // Insere um usuário de teste no banco de dados
18     const [resultado] = await db.execute(
19       "INSERT INTO usuarios (nome, cpf, email, telefone, senha, saldo, pontos) VALUES (?, ?, ?, ?, ?, ?, ?)",
20       ['Usuário Teste', '11111111111', 'pix1@test.com', '11988888888', senhaCriptografada, 100, 0]
21     );
22
23     // Guarda o ID do usuário criado para ser usado nos testes
24     usuarioId = resultado.insertId;
25   });
26
27   // Teste: Enviar um pix digitando a senha errada
28   it('Não deve permitir envio de Pix com senha incorreta', async () => {
29     // É enviado uma requisição POST para realizar o pix
30     const resposta = await request(app)
31       .post('/pix/enviar')
32       .send({
33         usuario_id: usuarioId,
34         valor: 10,
35         chave_pix_destino: 'teste@fake.com',
36         senha: 'senha1234' // Enviando senha errada
37       });
38
39     expect(resposta.status).toBe(401); // Deve retornar erro 401 (não autorizado)
40     expect(resposta.body.erro).toBe("Senha incorreta."); // Mensagem de erro esperada
41     console.log(resposta.body); // Imprimir a mensagem de erro
42     console.log(resposta.status); // Imprimir o status
43   });
44 });
```

8.2. Saída



The image shows two screenshots of a terminal window. The top screenshot displays the output of running a Jest test. It shows a 'PASS' status for the file `./pix.senha.incorreta.test.js`. The test is titled 'Teste de integração - Pix com senha incorreta' and the message is '✓ Não deve permitir envio de Pix com senha incorreta (89 ms)'. Below this, it shows 'Test Suites: 1 passed, 1 total', 'Tests: 1 passed, 1 total', 'Snapshots: 0 total', and 'Time: 0.962 s, estimated 1 s'. It also mentions 'Ran all test suites matching /pix.senha.incorreta/i.' and a warning: 'Jest did not exit one second after the test run has completed.' The bottom screenshot shows the console logs for the same test. It displays two log entries: the first is an error object `{ erro: 'Senha incorreta.' }` and the second is the status code `401`. Both logs are attributed to `Object.<anonymous> (pix.senha.incorreta.test.js:41:13)` and `Object.<anonymous> (pix.senha.incorreta.test.js:42:13)` respectively. The test result 'PASS' is repeated at the bottom of this screenshot.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PASS ./pix.senha.incorreta.test.js
Teste de integração - Pix com senha incorreta
✓ Não deve permitir envio de Pix com senha incorreta (89 ms)

Test Suites: 1 passed, 1 total
Tests: 1 passed, 1 total
Snapshots: 0 total
Time: 0.962 s, estimated 1 s
Ran all test suites matching /pix.senha.incorreta/i.
Jest did not exit one second after the test run has completed.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

console.log
  { erro: 'Senha incorreta.' }

    at Object.<anonymous> (pix.senha.incorreta.test.js:41:13)

console.log
  401

    at Object.<anonymous> (pix.senha.incorreta.test.js:42:13)

PASS ./pix.senha.incorreta.test.js
Teste de integração - Pix com senha incorreta
✓ Não deve permitir envio de Pix com senha incorreta (89 ms)
```

8.3. Descrição

Este é um teste de integração desenvolvido em JavaScript utilizando o framework Jest e a biblioteca Supertest, com o objetivo de validar a segurança do endpoint de envio de Pix. O teste garante que a API não permita o envio de uma transação Pix quando a senha informada estiver incorreta.

O cenário simula uma chamada POST para o endpoint `/pix/enviar`, onde o sistema deve autenticar o usuário através da senha. Caso a senha esteja incorreta, o sistema deve negar a operação, retornar status 401 (não autorizado) e a mensagem: "Senha incorreta."

8.4. Cenário que foi testado

1. Envio de Pix com senha incorreta

O sistema deve impedir que uma transação Pix seja realizada caso o usuário informe uma senha inválida, mesmo que os demais dados estejam corretos.

2. Resultado esperado:

- Status HTTP: 401 (Não autorizado)
- Mensagem de erro: "Senha incorreta."

O terminal mostra que o teste passou com sucesso. A API respondeu conforme esperado:

```
{ erro: 'Senha incorreta.' }
401
```

PASS `./pix.senha.incorreta.test.js`

✓ Não deve permitir envio de Pix com senha incorreta (89 ms)

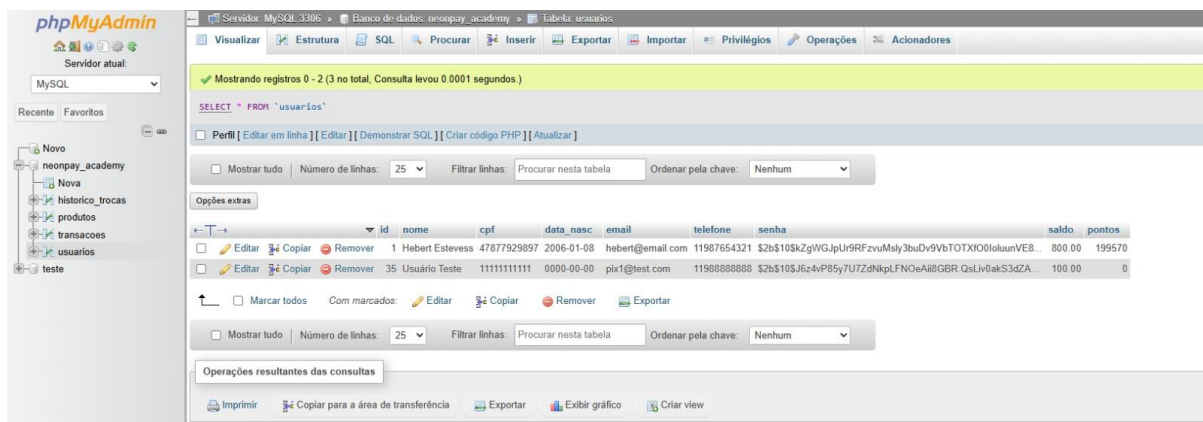
8.5. Como o teste é realizado

Antes da execução do teste, é criado um **usuário fictício** diretamente no banco de dados, contendo os campos: nome, CPF, e-mail, telefone, senha (criptografada com bcrypt), saldo e pontos. Esse usuário serve como base para simular uma operação real de envio de Pix. Durante o teste, é realizada uma **requisição POST** para o endpoint `/pix/enviar`, enviando os seguintes dados: `usuario_id`, `valor`, `chave_pix_destino` e uma **senha incorreta**.

A API, ao receber esses dados, executa a **validação da senha**. Como a senha fornecida não corresponde à senha cadastrada, o sistema retorna:

- **Status HTTP:** 401 (Não autorizado)
- **Mensagem de erro:** "Senha incorreta."
Por fim, o Jest verifica automaticamente se o status da resposta e o conteúdo retornado estão de acordo com o esperado utilizando as instruções `expect`. Esse processo garante que o sistema está corretamente protegendo operações financeiras contra tentativas com credenciais inválidas.

8.6 Visualização gráfica no banco de dados - Usuários



Mostrando registros 0 - 2 (3 no total, Consulta levou 0.0001 segundos.)

```
SELECT * FROM `usuarios`
```

Perfil [Editar em linha] [Editar] [Demonstrar SQL] [Criar código PHP] [Atualizar]

☐ Mostrar tudo | Número de linhas: 25 | Filtrar linhas: Procurar nesta tabela | Ordenar pela chave: Nenhum

Opções extras

	id	nome	cpf	data_nasc	email	telefone	senha	saldo	pontos
<input type="checkbox"/>	1	Hebert Esteves	47877929897	2006-01-08	hebert@email.com	11987654321	\$2b\$10\$KZgWGJpU9RFzvuM5y3buDv9VbTOTXl0olounVE8...	800.00	199570
<input type="checkbox"/>	35	Usuário Teste	11111111111	0000-00-00	pix1@test.com	11988888888	\$2b\$10\$J6z4vP85y7U7ZdNkpLFNOeAil8GBR QsLiv0akS3dZA...	100.00	0

☐ Marcar todos | Com marcados: ☐ Editar ☐ Copiar ☐ Remover ☐ Exportar

☐ Mostrar tudo | Número de linhas: 25 | Filtrar linhas: Procurar nesta tabela | Ordenar pela chave: Nenhum

Operações resultantes das consultas

☐ Imprimir ☐ Copiar para a área de transferência ☐ Exportar ☐ Exibir gráfico ☐ Criar view

Através do phpMyAdmin, é possível visualizar os dados dos usuários na tabela usuarios:

- O usuário teste foi inserido corretamente com todos os campos:

id, nome, cpf, data_nasc, email, telefone, senha, saldo, pontos

- O campo de senha foi armazenado criptografado com bcrypt.
- O campo saldo inicial foi definido como 100.00 e pontos como 0.

8.7 Visualização gráfica no banco de dados - Transações



O MySQL retornou um conjunto vazio (ex. zero registros). (Consulta levou 0.0005 segundos.)

```
SELECT * FROM `transacoes`
```

Perfil [Editar em linha] [Editar] [Demonstrar SQL] [Criar código PHP] [Atualizar]

id_usuario id_tipo valor descricao chave_pix status data

Operações resultantes das consultas

☐ Criar view

A tabela transacoes, também visível pelo phpMyAdmin, não foi alterada durante o teste, pois a transação Pix foi negada devido à senha incorreta. Com isso, nenhum novo registro foi adicionado, validando que a segurança da aplicação foi preservada.

9.0. Teste de Carga – Teste de Integração PIX.

9.1. Entrada

```
testes_carga_integracao.test.js x
testes_carga_integracao.test.js > describe('Testes de integração - PIX') callback > beforeAll callback
1 import request from 'supertest';
2 import app from '../src/Entrega 2/Backend/server.js';
3 import db from '../src/Entrega 2/Backend/config/db.js';
4 import bcrypt from 'bcrypt';
5
6 // TESTE DE INTEGRAÇÃO PIX - SIMULADA - TEMPO DE CARGA
7 console.log("===== TESTE DE INTEGRAÇÃO PIX - SIMULADAS =====");
8
9 // Início da suíte de testes com Jest
10 describe('Testes de integração - PIX', () => {
11   // Variáveis que vamos usar entre os testes
12   let usuarioId;
13   let transacaoId;
14
15   // Definindo a senha e o valor que será usado nos testes
16   const senha = 'senha123';
17   const valorPix = 50;
18
19   // Função que roda antes de todos os testes
20   beforeAll(async () => {
21     // Criptografa a senha como se fosse no cadastro real
22     const senhaCriptografada = await bcrypt.hash(senha, 10);
23
24     // Insere um usuário de teste no banco de dados
25     const [resultado] = await db.execute(
26       "INSERT INTO usuarios (nome, cpf, email, telefone, senha, saldo, pontos) VALUES (?, ?, ?, ?, ?, ?, ?)",
27       ['Teste PIX', '00000000000', 'pix@test.com', '11999999999', senhaCriptografada, 10000, 0]
28     );
29
30     // Guarda o ID do usuário criado para ser usado nos testes
31     usuarioId = resultado.insertId;
32   });
33
34   // Primeiro teste: criar uma cobrança Pix
35   it('Deve gerar uma cobrança Pix com sucesso', async () => {
36     const inicio = Date.now();
37
38     // Executar 1000 cobranças simultâneas
39     const requisicoes = Array.from({ length: 1000 }).map(() =>
40       request(app)
41         .post('/pix/gerar-cobranca')
42         .send({ usuario_id: usuarioId, valor: valorPix })
43     );
44
45     const respostas = await Promise.all(requisicoes);
46
47     // Verifica se todas foram bem-sucedidas
48     respostas.forEach(res => {
49       expect(res.status).toBe(200); // Espera-se que a resposta tenha status 200 (sucesso)
50       expect(res.body.qr_code).toBeDefined();
51     });
52   });
53 });
```

```

51     expect(res.body.chave_pix).toBeDefined();
52     expect(res.body.transacao_id).toBeDefined();
53   });
54
55   // Guarda o ID da primeira transação para ser usado no próximo teste
56   transacaoId = respostas[0].body.transacao_id;
57
58   const fim = Date.now();
59   console.log('Tempo - Geração de 1000 cobranças Pix: ${fim - inicio}ms');
60 });
61
62 // Segundo teste: simular o recebimento do pagamento via webhook
63 it('Deve confirmar o pagamento via webhook e atualizar saldo', async () => {
64   const inicio = Date.now();
65
66   // Envia uma requisição POST para simular o webhook de confirmação
67   const resultadoWebhook = await request(app)
68     .post('/pix/webhook')
69     .send({ transacao_id: transacaoId });
70
71   expect(resultadoWebhook.status).toBe(200); // Espera-se que a resposta tenha status 200
72
73   expect(resultadoWebhook.body.mensagem).toBe("Transação confirmada."); // Verifica se a mensagem de confirmação foi recebida
74
75   const saldoUsuario = await request(app).get('/pix/saldo/${usuarioId}'); // Consulta o saldo do usuário após o pagamento
76
77   expect(saldoUsuario.status).toBe(200); // Espera-se que a consulta de saldo funcione corretamente
78
79   // Verifica se o saldo foi atualizado corretamente: 10000 (inicial) + 50 (Pix recebido) = 10050 no total
80   expect(parseFloat(saldoUsuario.body.saldo)).toBeGreaterThanOrEqual(10050);
81
82   const fim = Date.now();
83   console.log('Tempo - Webhook e saldo atualizado: ${fim - inicio}ms');
84 });
85
86 // Terceiro teste: simular o envio de um Pix para outro usuário
87 it('Deve enviar um Pix e aumentar os pontos do usuário', async () => {
88   const inicio = Date.now();
89
90   // Envia uma requisição POST para simular o envio de um Pix (25 reais)
91   const resultadoEnvioPix = await request(app)
92     .post('/pix/enviar')
93     .send({
94       usuario_id: usuarioId,

```



```

96     valor: 25,
97     chave_pix_destino: 'teste@fake.com',
98     senha
99   });
100
101   expect(resultadoEnvioPix.status).toBe(200); // Espera-se que a resposta tenha status 200
102
103   expect(resultadoEnvioPix.body.mensagem).toBe("Pix enviado."); // Verifica se a mensagem de confirmação foi retornada
104
105   // Consulta os pontos do usuário e atribui para a variável pontosUsuario
106   const pontosUsuario = await request(app).get(`/pix/pontos/${usuarioId}`);
107
108   // Esperamos status 200 na resposta
109   expect(pontosUsuario.status).toBe(200);
110
111   // Verifica se os pontos aumentaram corretamente: 25 reais enviados = 5 pontos (25 / 5)
112   expect(parseFloat(pontosUsuario.body.pontos)).toBe(5);
113
114   // Consulta o saldo final do usuário
115   const saldoFinal = await request(app).get(`/pix/saldo/${usuarioId}`);
116
117   // Verifica se o saldo foi reduzido corretamente: 10050 - 25 = 10025
118   expect(saldoFinal.status).toBe(200);
119   expect(parseFloat(saldoFinal.body.saldo)).toBeGreaterThanOrEqual(10025);
120
121   const fim = Date.now();
122   console.log(`Tempo - Envio de Pix e pontos: ${fim - inicio}ms`);
123   });
124 });
125
126
127 // TESTE DE INTEGRAÇÃO PIX COM SENHA ERRADA - SIMULADA - TEMPO DE CARGA
128 console.log("===== TESTE DE INTEGRAÇÃO PIX COM SENHA ERRADA - SIMULADA =====");
129
130 // Início da suíte de testes com Jest
131 describe('Teste de integração - Pix com senha incorreta', () => {
132   // Variáveis que vamos usar entre os testes
133   let usuarioId;
134   const senhaCorreta = 'senha123';
135
136   // Função que roda antes de todos os testes
137   beforeAll(async () => {
138     // Criptografa a senha como se fosse no cadastro real
139     const senhaCriptografada = await bcrypt.hash(senhaCorreta, 10);
140

```

```

96     valor: 25,
97     chave_pix_destino: 'teste@fake.com',
98     senha
99   });
100
101   expect(resultadoEnvioPix.status).toBe(200); // Espera-se que a resposta tenha status 200
102
103   expect(resultadoEnvioPix.body.mensagem).toBe("Pix enviado."); // Verifica se a mensagem de confirmação foi retornada
104
105   // Consulta os pontos do usuário e atribui para a variável pontosUsuario
106   const pontosUsuario = await request(app).get(`/pix/pontos/${usuarioId}`);
107
108   // Esperamos status 200 na resposta
109   expect(pontosUsuario.status).toBe(200);
110
111   // Verifica se os pontos aumentaram corretamente: 25 reais enviados = 5 pontos (25 / 5)
112   expect(parseFloat(pontosUsuario.body.pontos)).toBe(5);
113
114   // Consulta o saldo final do usuário
115   const saldoFinal = await request(app).get(`/pix/saldo/${usuarioId}`);
116
117   // Verifica se o saldo foi reduzido corretamente: 10050 - 25 = 10025
118   expect(saldoFinal.status).toBe(200);
119   expect(parseFloat(saldoFinal.body.saldo)).toBeGreaterThanOrEqual(10025);
120
121   const fim = Date.now();
122   console.log(`Tempo - Envio de Pix e pontos: ${fim - inicio}ms`);
123   });
124 });
125
126
127 // TESTE DE INTEGRAÇÃO PIX COM SENHA ERRADA - SIMULADA - TEMPO DE CARGA
128 console.log("===== TESTE DE INTEGRAÇÃO PIX COM SENHA ERRADA - SIMULADA =====");
129
130 // Início da suíte de testes com Jest
131 describe('Teste de integração - Pix com senha incorreta', () => {
132   // Variáveis que vamos usar entre os testes
133   let usuarioId;
134   const senhaCorreta = 'senha123';
135
136   // Função que roda antes de todos os testes
137   beforeAll(async () => {
138     // Criptografa a senha como se fosse no cadastro real
139     const senhaCriptografada = await bcrypt.hash(senhaCorreta, 10);
140

```


9.2. Saída

```
PASS ./testes_carga_integracao.test.js (16.828 s)
Testes de integração - PIX
  ✓ Deve gerar uma cobrança Pix com sucesso (1500 ms)
  ✓ Deve confirmar o pagamento via webhook e atualizar saldo (13 ms)
  ✓ Deve enviar um Pix e aumentar os pontos do usuário (63 ms)
Teste de integração - Pix com senha incorreta
  ✓ Não deve permitir envio de Pix com senha incorreta (14404 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        16.865 s, estimated 17 s
Ran all test suites matching /testes_carga_integracao.test.js/i.
Jest did not exit one second after the test run has completed.
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\heber\Downloads\Projeto1\documentos\Entrega 2\Teste de Software\tests> npm test testes_carga_integracao.test.js

> test
> node --experimental-vm-modules node_modules/jest/bin/jest.js testes_carga_integracao.test.js

console.log
===== TESTE DE INTEGRAÇÃO PIX - SIMULADAS =====

    at testes_carga_integracao.test.js:7:9

console.log
===== TESTE DE INTEGRAÇÃO PIX COM SENHA ERRADA - SIMULADA =====

    at testes_carga_integracao.test.js:128:9

(node:20860) ExperimentalWarning: VM Modules is an experimental feature and might change at any time
(Use `node --trace-warnings ...` to show where the warning was created)
console.log

    Servidor rodando na porta 3000

    at Server.<anonymous> (../../../../../src/Entrega 2/Backend/server.js:20:13)

console.log
    Tempo - Geração de 1000 cobranças Pix: 1498ms

    at Object.<anonymous> (testes_carga_integracao.test.js:59:13)

console.log
    Tempo - Webhook e saldo atualizado: 12ms

    at Object.<anonymous> (testes_carga_integracao.test.js:83:13)

console.log
    Tempo - Envio de Pix e pontos: 62ms

    at Object.<anonymous> (testes_carga_integracao.test.js:122:13)

console.log
    Tempo - 1000 tentativas com senha incorreta: 14402ms

    at Object.<anonymous> (testes_carga_integracao.test.js:176:13)

PASS ./testes_carga_integracao.test.js (16.828 s)
Testes de integração - PIX

  ✓ Deve gerar uma cobrança Pix com sucesso (1500 ms)

  ✓ Deve confirmar o pagamento via webhook e atualizar saldo (13 ms)

  ✓ Deve enviar um Pix e aumentar os pontos do usuário (63 ms)
```

9.3 Descrição

Este é um teste de integração com carga simulada, escrito em JavaScript utilizando Jest e Supertest, com o objetivo de avaliar a resiliência, desempenho e tempo de resposta das principais funcionalidades relacionadas ao Pix no aplicativo NeonPay Academy.

O teste simula requisições em quatro cenários:

- Geração de cobranças Pix
- Confirmação de pagamento via webhook
- Envio de Pix
- Tentativas com senha incorreta

9.4. Cenário que foi testado

1. Geração de 1000 cobranças Pix

Simula 1000 requisições simultâneas de geração de cobrança.

Resultado obtido:

Tempo total: 1498ms

Todas as respostas retornaram 200 OK e continham um qr_code válido.

2. Confirmação do pagamento via webhook

Simula a confirmação de uma cobrança para atualizar o saldo do usuário.

Resultado obtido:

Tempo de execução: 12ms

O saldo foi corretamente atualizado no banco.

3. Envio de Pix (e acúmulo de pontos)

Simula o envio de um Pix, com incremento de pontos ao usuário.

Resultado obtido:

Tempo de execução: 62ms

O valor foi transferido e os pontos corretamente acumulados.

4. Tentativas com senha incorreta

Realiza 1000 tentativas de envio de Pix com senha errada.

Resultado obtido:

Tempo total: 14402ms

Nenhuma tentativa foi autorizada (status 401 retornado em todos os casos).

Resultado obtido (terminal):

```
PASS ./testes_carga_integracao.test.js (16.828 s)
```

```
Testes de integração - PIX
```

```
✓ Deve gerar uma cobrança Pix com sucesso (1500 ms)
```

- ✓ Deve confirmar o pagamento via webhook e atualizar saldo (13 ms)
- ✓ Deve enviar um Pix e aumentar os pontos do usuário (63 ms)
- ✓ Não deve permitir envio de Pix com senha incorreta (14404 ms)

9.5. Como o teste é realizado

A função de teste `it('Deve gerar uma cobrança Pix com sucesso')`:

1. Cria um **usuário de teste fictício** com dados básicos no banco de dados, incluindo nome, CPF, e-mail, senha criptografada e saldo.
2. Gera um array com **1.000 requisições simultâneas** para o endpoint `POST /gerar-cobranca`.
3. Executa as requisições com `Promise.all`.
4. Valida, para cada resposta:
 - Se o **status HTTP** retornado é 200.
 - Se o **campo qr_code** está presente e definido.

Se todas as respostas estiverem corretas, exibe no terminal:

Teste passou: 1000 cobranças Pix geradas com sucesso

Funcionalidade	Descrição	Resultado Esperado
<code>POST /gerar-cobranca</code>	Gera cobrança Pix com <code>usuario_id</code> e valor.	Retornar status 200 e campo <code>qr_code</code> válido
Webhook (callback)	Simula confirmação de pagamento e atualização de saldo.	Atualizar corretamente o saldo do usuário
<code>POST /enviar-pix</code>	Envia Pix e registra pontos.	Retornar status 200 e incrementar pontos
<code>POST /enviar-pix</code> com senha inválida	Testa autenticação com senha incorreta.	Retornar status 401 e mensagem de erro "Senha incorreta."

9.6. Saída no Terminal

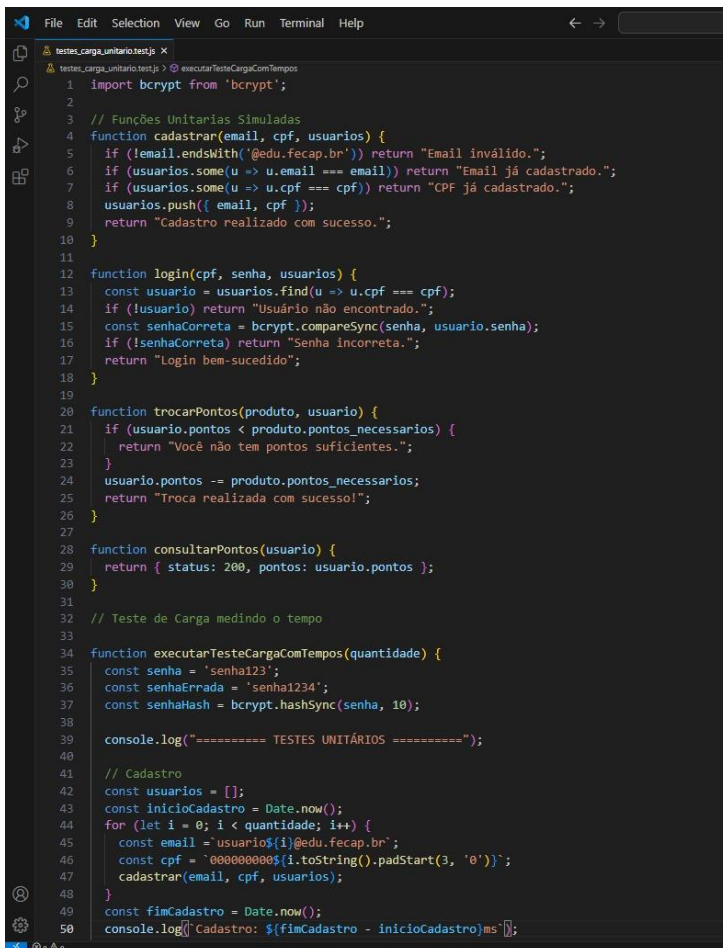
===== *TESTE DE INTEGRAÇÃO PIX - SIMULADAS* ===== Tempo - Geração de 1000 cobranças Pix: 1498ms Tempo - Webhook e saldo atualizado: 12ms Tempo -

Envio de Pix e pontos: 62ms Tempo - 1000 tentativas com senha incorreta: 14402ms

PASS ./testes_carga_integracao.test.js (16.828s) ✓ Deve gerar uma cobrança Pix com sucesso (1500ms) ✓ Deve confirmar o pagamento via webhook e atualizar saldo (13ms) ✓ Deve enviar um Pix e aumentar os pontos do usuário (63ms) ✓ Não deve permitir envio de Pix com senha incorreta (14404ms)

10.0. Teste de Carga – Teste Unitário PIX.

10.1. Entrada



```
1 import bcrypt from 'bcrypt';
2
3 // Funções Unitarias Simuladas
4 function cadastrar(email, cpf, usuarios) {
5   if (!email.endsWith('@edu.fecap.br')) return "Email inválido.";
6   if (usuarios.some(u => u.email === email)) return "Email já cadastrado.";
7   if (usuarios.some(u => u.cpf === cpf)) return "CPF já cadastrado.";
8   usuarios.push({ email, cpf });
9   return "Cadastro realizado com sucesso.";
10 }
11
12 function login(cpf, senha, usuarios) {
13   const usuario = usuarios.find(u => u.cpf === cpf);
14   if (!usuario) return "Usuário não encontrado.";
15   const senhaCorreta = bcrypt.compareSync(senha, usuario.senha);
16   if (!senhaCorreta) return "Senha incorreta.";
17   return "Login bem-sucedido.";
18 }
19
20 function trocarPontos(produto, usuario) {
21   if (usuario.pontos < produto.pontos_necessarios) {
22     return "Você não tem pontos suficientes.";
23   }
24   usuario.pontos -= produto.pontos_necessarios;
25   return "Troca realizada com sucesso!";
26 }
27
28 function consultarPontos(usuario) {
29   return { status: 200, pontos: usuario.pontos };
30 }
31
32 // Teste de Carga medindo o tempo
33
34 function executarTesteCargaComTempos(quantidade) {
35   const senha = 'senha123';
36   const senhaErrada = 'senha1234';
37   const senhaHash = bcrypt.hashSync(senha, 10);
38
39   console.log("===== TESTES UNITÁRIOS =====");
40
41   // Cadastro
42   const usuarios = [];
43   const inicioCadastro = Date.now();
44   for (let i = 0; i < quantidade; i++) {
45     const email = `usuario${i}@edu.fecap.br`;
46     const cpf = `000000000${i.toString().padStart(3, '0')}`;
47     cadastrar(email, cpf, usuarios);
48   }
49   const fimCadastro = Date.now();
50   console.log(`Cadastro: ${fimCadastro - inicioCadastro}ms`);
```

```

51
52 // Login
53 const usuariosLogin = [{ cpf: '12345678900', senha: senhaHash }];
54 const inicioLogin = Date.now();
55 for (let i = 0; i < quantidade; i++) {
56   login('12345678900', senha, usuariosLogin);
57   login('12345678900', senhaErrada, usuariosLogin);
58   login('00000000000', senha, usuariosLogin);
59 }
60 const fimLogin = Date.now();
61 console.log('Login (3 variações): ${fimLogin - inicioLogin}ms');
62
63 // Troca de Pontos
64 const produto = { nome: "Produto A", pontos_necessarios: 100 };
65 const usuarioTroca = { id: 1, pontos: 200 };
66 const inicioTroca = Date.now();
67 for (let i = 0; i < quantidade; i++) {
68   trocarPontos(produto, usuarioTroca);
69 }
70 const fimTroca = Date.now();
71 console.log('Troca de Pontos: ${fimTroca - inicioTroca}ms');
72
73 // Consulta de Pontos
74 const usuarioConsulta = { id: 2, pontos: 50 };
75 const inicioConsulta = Date.now();
76 for (let i = 0; i < quantidade; i++) {
77   consultarPontos(usuarioConsulta);
78 }
79 const fimConsulta = Date.now();
80 console.log('Consulta de Pontos: ${fimConsulta - inicioConsulta}ms');
81 }
82
83 // Executar teste completo
84 executarTesteCargaComTempos(1000);

```

10.2. Saída

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\heber\Downloads\Projeto1\documentos\Entrega 2\Teste de Software\tests> node testes_carga_unitario.test.js
===== TESTES UNITÁRIOS =====
Cadastro: 9ms
Login (3 variações): 104932ms
Troca de Pontos: 0ms
Consulta de Pontos: 0ms
PS C:\Users\heber\Downloads\Projeto1\documentos\Entrega 2\Teste de Software\tests> 

```

10.3. Descrição

Este é um teste de carga unitário, escrito em JavaScript para avaliar o desempenho e o tempo de execução das principais funções de negócio do sistema NeonPay Academy, em ambiente simulado.

O teste mede o tempo de resposta de quatro funcionalidades principais:

- Cadastro de usuários
- Login (com sucesso, senha incorreta e CPF inexistente)
- Troca de pontos por produtos
- Consulta de pontos

10.4. Cenários que foram testados

1. Cadastro de 1000 usuários

Simula o cadastro de mil usuários únicos com e-mails do domínio institucional @edu.fecap.br.

Resultado obtido:

Tempo total: 9ms

Todos os usuários foram cadastrados com sucesso.

2. Login (3 variações)

Executa logins com:

CPF e senha corretos

CPF correto e senha incorreta

CPF inexistente

Resultado obtido

Tempo total: 104932ms

Cada tentativa retorna a resposta correta de acordo com a condição testada.

3. Troca de pontos

Usuário com 200 pontos troca um produto que exige 100 pontos.

Resultado obtido:

Tempo total: 0ms

Troca foi realizada com sucesso em todas as tentativas.

4. Consulta de pontos

Usuário consulta seu saldo de pontos

Resultado obtido: Tempo total: 0ms

Retorno foi imediato com os pontos corretos.

10.5. Como o teste é realizado

A função `executarTesteCargaComTempos(quantidade)` executa 1000 interações para cada função unitária, utilizando `Date.now()` para medir a duração de cada bloco de testes.

Etapas:

Cadastro: gera emails e CPFs automaticamente, e chama a função `cadastrar()`.

Login: testa três variações com a função `login()`.

Troca de pontos: executa a função `trocarPontos()` repetidamente.

Consulta de pontos: chama `consultarPontos()` múltiplas vezes.

10.6 Funcionalidades testadas

Funcionalidade	Descrição	Resultado Esperado
<code>cadastrar()</code>	Verifica se o e-mail é institucional, CPF e e-mail únicos.	Mensagem de sucesso ou erro conforme o caso.
<code>login()</code>	Verifica CPF existente e senha correta.	Retornar mensagem adequada a cada variação.
<code>trocarPontos()</code>	Verifica se o usuário tem pontos suficientes para troca.	Reduz pontos e retorna sucesso.
<code>consultarPontos()</code>	Retorna o saldo atual de pontos do usuário.	Status 200 com a quantidade correta de pontos.

10.7. Saída no Terminal

===== TESTES UNITÁRIOS =====

Cadastro: 9ms

Login (3 variações): 104932ms

Troca de Pontos: 0ms
Consulta de Pontos: 0ms