1. **Cenário geral do Sistema de Login**

O sistema de login de um site é fundamental para plataformas de e-commerce, onde permite que os usuários possam se autenticar, acessar suas contas e gerenciar os seus dados dentro da plataforma.

As principais funcionalidades são:
1. Autenticação: Responsável por verificar as credenciais dos usuários.
2. Recuperação de senha: Responsável por recuperar a senha dos usuários caso ocorra a perda.
3. Sessões: Responsável por armazenar os dados do usuário durante a navegação do mesmo na plataforma.

Casos de teste:
1. Verificar se é possível se autenticar com diferentes credenciais.
2. Verificar se ao recuperar uma senha, é possível se autenticar com as novas credenciais.
3. Verificar se ao se autenticar, os dados estão sendo armazenados na sessão.

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

As técnicas que serão utilizadas, serão as seguintes:

Teste Funcional
1. Fazer login com usuário e senha válido
2. Recuperar a senha com sucesso
3. Verificar se os dados estão sendo armazenados na sessão

Teste Estrutural
1. Verificar se ao fazer login com credenciais incorretas, uma exceção será lançada corretamente.
2. Verificar se ao recuperar uma senha com usuário incorreto, uma exceção será lançada corretamente.

Para o desenvolvimento dos testes, foram utilizadas as seguintes ferramentas.

**NestJS**

NestJS é um framework Node.js de código aberto destinado ao desenvolvimento de aplicativos do lado do servidor. Foi criado por Kamil Mysliwiec e lançado em 2017. Sob o capô, por padrão, o NestJS faz uso do framework Express.js, sendo também compatível com o Fastify. Sua arquitetura é fortemente inspirada no Angular.

**SQLite**

SQLite é uma biblioteca em linguagem C que implementa uma base de dados SQL embutida. Programas que usem a biblioteca SQLite podem ter acesso a banco de dados SQL sem executar um processo SGBD separado

**Jest**

Jest é uma estrutura de teste de JavaScript construída sobre o Jasmine e mantida pela Meta. Ele foi projetado e construído por Christoph Nakazawa com foco na simplicidade e suporte para grandes aplicativos da web. Funciona com projetos usando Babel, TypeScript, Node.js, React, Angular, Vue.js e Svelte.

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

Caso de teste de login com dados válidos

- ❖ Dados de entrada
  - ➢ email: “user@test.com”
  - ➢ password: “pwd123”

Resultado esperado

- Usuário recebe um token da sessão criada

Caso de teste de login com dados inválidos

- ❖ Dados de entrada
  - ➢ email: “user@test.com”
  - ➢ password: “pwd12345”

Resultado esperado

- Usuário deve receber um alerta informando que as credenciais informadas são inválidas

Caso de teste de recuperação de senha

- ❖ Dados de entrada
  - ➢ email: “user@test.com”
  - ➢ password:”pwd123”
  - ➢ new password: “pwd1234”

Resultado esperado

- O usuário deve receber um alerta informando que sua senha foi alterada com sucesso.

Caso de teste de recuperação de senha com usuario invalido

- Dados de entrada
  - email: "user@test.com"
  - password:"pwd1234567"
  - new password: "pwd1234"

Resultado esperado

- O usuário deve receber um alerta informando que a senha atual é inválida.

4. **Execução (quando e como será testado)**

Para iniciar os testes, implementamos a função "beforeAll", que irá criar um usuário com email e senha para nosso caso de teste:

```
beforeAll(async()=>{
            const password = await bcrypt.hash('pwd123',10);
            await userRepository.save(
                userRepository.create({
                    id: userId,
                    email: 'user@test.com',
                    password: password
                })
            )
        });
```

Para validar os casos, utilizamos a função "it", que é onde definimos o primeiro caso de teste Neste teste, fizemos uma requisição para nosso backend passando as credenciais do usuário que criamos.

O método "expect" permite que verifiquemos alguns resultados para validar o teste.
Esta função valida se a requisição foi aceita, depois verifica se um token de sessão foi recebido, e verifica se dentro do token está o usuário que passamos.

```
it('POST /login | Positive -> should be able to allow user access and receved a session token', async () => {

            const response = await testModule.httpRequest
                .post('/login')
                .send({
                    email: 'user@test.com',
                    password: 'pwd123'
                });

            expect(response.status).toBe(HttpStatus.ACCEPTED)

            const sessionToken = response.text;

            expect(sessionToken).toBeDefined();


            const userToSession = JSON.stringify({
                user: {
                    email: 'user@test.com'
                }
            })
            const comparisonBetweenTokens =  await bcrypt.compare(userToSession, sessionToken)

            expect(comparisonBetweenTokens ).toBe(true);

            session = sessionToken;

        });
```

Segue  todos os testes feitos para este sistema

Verifique se é possível fazer login com as credenciais corretas.

```
it('POST /login | Positive -> should be able to allow user access and
received a session token', async () => {

            const response = await testModule.httpRequest
                .post('/login')
                .send({
                    email: 'user@test.com',
                    password: 'pwd123'
                });

            expect(response.status).toBe(HttpStatus.ACCEPTED)

            const sessionToken = response.text;

            expect(sessionToken).toBeDefined();


            const userToSession = JSON.stringify({
                user: {
                    email: 'user@test.com'
                }
            })
            const comparisonBetweenTokens =  await
bcrypt.compare(userToSession, sessionToken)

            expect(comparisonBetweenTokens ).toBe(true);

            session = sessionToken;

        });
```

Verifica se ao tentar fazer login com uma senha inválida, uma mensagem de erro é recebida.

```
it('POST /login | Negative -> should get an error message saying that
the password is invalid', async () => {

            const response = await testModule.httpRequest
                .post('/login')
                .send({
                    email: 'user@test.com',
                    password: 'pwd12345'
                });

            expect(response.status).toBe(HttpStatus.ACCEPTED)

            const result = response.body as { error: { message : string
} }

            expect(result.error.message).toBe('Senha invalida');

        });
```

Verifica se ao tentar recuperar a senha passando uma senha inválida, uma mensagem de erro é recebida.

```
it('POST /reset-password | Negative -> should be able to change the
password', async () => {

            const response = await testModule.httpRequest
                .post('/reset-password')
                .send({
                    email: 'user@test.com',
                    password: 'pwd12345',
                    newPassword: 'pwd1234'
                });

            expect(response.status).toBe(HttpStatus.ACCEPTED)

            const result = response.body as { error: { message : string
} }

            expect(result.error.message).toBe('Senha invalida');

        });
```

Verifica se é possível recuperar a senha fornecendo as credenciais corretas

```
it('POST /reset-password | Positive -> should be able to change the
password', async () => {

            const response = await testModule.httpRequest
                .post('/reset-password')
                .send({
                    email: 'user@test.com',
                    password: 'pwd123',
                    newPassword: 'pwd1234'
                });

            expect(response.status).toBe(HttpStatus.ACCEPTED)

            const result = response.body as {passwordChange: boolean};

            expect(result.passwordChange).toBe(true);

        });
```

Verifica se é possivel fazer login na aplicação utilizando a nova senha

```
it('POST /login -> should be able to allow user access after change
password', async () => {

            const response = await testModule.httpRequest
                .post('/login')
                .send({
                    email: 'user@test.com',
                    password: 'pwd1234'
                });

            expect(response.status).toBe(HttpStatus.ACCEPTED)

            const sessionToken = response.text;

            expect(sessionToken).toBeDefined();

            const userToSession = JSON.stringify({
                user: {
                    email: 'user@test.com'
                }
            })
```

Verifica se é possível fazer logout na aplicação enviando o token da sessão

```
it('POST /logout -> should be able to log out user', async () => {

            expect(session).toBeDefined();

            const response = await testModule.httpRequest
                .post('/logout')
                .send({
                    token: session
                });

            expect(response.status).toBe(HttpStatus.ACCEPTED)

            const result = response.body as {logout: boolean};

            expect(result.logout).toBe(true);

        });
```

Resultado final dos testes

```
PASS tests-login test/auth/index.e2e-spec.ts (8.874 s)
  Auth
    /login
      √ POST /login | Positive -> should be able to allow user access and receved a session token (335 ms)
      √ POST /login | Negative -> should get an error message saying that the password is invalid (77 ms)
      √ POST /reset-password | Negative -> should be able to change the password (74 ms)
      √ POST /reset-password | Positive -> should be able to change the password (1381 ms)
      √ POST /login -> should be able to allow user access after change password (202 ms)
      √ POST /logout -> should be able to log out user (60 ms)

Test Suites: 1 passed, 1 total
Tests:       6 passed, 6 total
Snapshots:   0 total
Time:        9.003 s
Ran all test suites matching /.\\test\\auth/i.
Done in 10.12s.
```

Segue o link para o repositório do projeto no GitHub

- https://github.com/lucascandido-ti/nestjs-sql-jest-tests

## 5. Análise dos Resultados e próximos passos

Os testes são extremamente importantes para a qualidade do nosso software, principalmente falando de desenvolvimento Web.
Uma aplicação Web pode receber milhares de acessos, e se algum módulo não for devidamente testado, um grande problema pode aparecer, e muitos usuários deixaram de utilizar o software.