

UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO

CAMPUS PAU DOS FERROS

DISCIPLINA: TESTE DE SOFTWARE

DOCENTE: ALYSSON FILGUEIRA MILANEZ

ALUÍZIO AIRES DA SILVA JUNIOR

ANGELO SILVANO DA SILVA SABINO

LEVÍTICO RIMON PEREZ ANDRADE ALVES

LUDMILA COSTA MONTEIRO

MATEUS GOMES SALOME

Relatório de Testes de Caixa Branca e Testes de Caixa Preta

PAU DOS FERROS – RN

2025

**Testes de Caixa Branca**

Todos os testes foram feitos pelo Jest, usando o comando para rodar: ‘npm test’ e estão no repositório do git hub.

Testes:

taskUtils.test.js

- Função testada: formatarData(dataStr)

- A função recebe uma string no formato “YYYY-MM-DD” e retorna ‘DD/MM/YYYY’.

- Caso a string esteja vazia ou em formato errado, retorna uma string vazia ou resultado inesperado.

- Casos de teste:

1. Entrada válida: ‘2025-06-20’ Resultado esperado: ‘20/06/2025’.

2. Entrada vazia: “Resultado esperado”.

3. Entrada mal formatada: ‘20/06/2025’ resultado não deve ser ‘20/06/2025’

- Complexidade Ciclomática: 3.

- Todos os testes foram executados com sucesso.

validarLogin.test.js

- Função testada: validarLogin(email, senha).

- Verifica se a função retorna corretamente mensagens de erro quando os campos obrigatórios( email e senha) não são preenchidos, e se retorna “ok” quando ambos estão válidos.

- Casos de teste:

1. Email e senha preenchidos corretamente. Resultado esperado: “ok”.

2. Email vazio. Resultado esperado: “Email obrigatório”.

3. Senha vazia. Resultado esperado: “Senha obrigatória”.

- Complexidade Ciclomática: 3.

- Todos os testes foram executados com sucesso.

authMiddleware.test.js

- Função testada: garante que apenas usuários autênticos com token JWT tenham acesso à rotas protegidas.

- Verifica o cabeçalho Authorization de uma requisição HTTP, valida o token JWT recebido e, caso válido, insere as informações do usuário decodificadas em `req.user`, permitindo a continuidade da requisição via `next()`.

- Casos de teste:

1. Sem token no cabeçalho: deve retornar status 401 com mensagem ‘Acesso negado. Nenhum token fornecido’.

2. Token inválido: deve retornar status 400 com mensagem ‘Token inválido’.

3. Token válido: deve chamar `next()` e definir `req.user` com os dados decodificados.

- Complexidade Ciclomática: 3.

- Todos os testes foram executados com sucesso.

validarCamposTarefa.test.js

- Função testada: titulo, data.

- A função verifica se os campos de uma tarefa estão preenchidos corretamente, retorna mensagem de erro se o titulo ou a data estiverem vazios e retorna ‘ok’ se ambos os campos estiverem preenchidos.

- Casos de teste:

1. Entrada válida: titulo = ‘Estudar’, data = ‘2025-06-21’. Resultado: ‘ok’.

2. Titulo vazio: titulo = ‘’, data = ‘2025-06-21’. Resultado: ‘Titulo obrigatorio’.

3. Data vazia: titulo = ‘Estudar’, data = ‘’. Resultado: ‘Data obrigatoria’.

- Complexidade Ciclomática: 3.

- Todos os testes foram executados com sucesso.

filtrarTarefas.test.js

- Função testada: status, tarefa.

- A função filtrarTarefasPorStatus recebe dois parâmetros: um status (“pendende” ou “feita”) e um array de tarefas. Ela retorna um novo array contendo apenas as tarefas cujo status corresponde ao filtro fornecido.

- Casos de teste:

1. Deve retornar apenas tarefas com status “pendente”.

2. Deve retornar apenas tarefas com status “feita”.

3. Deve retornar array vazio se não houver tarefas com status fornecido.

4. Deve retornar array vazio se o array de tarefas estiver vazio.

- Complexidade Ciclomática: 1.

- Todos os testes foram executados com sucesso.

toggleTheme.test.js

- Função testada: dark-mode.

- Garante que a função adicione a classe ‘dark-mode’ ao body se estiver ausente, garante remover a classe ‘dark-mode’ se já estiver presente e atualiza corretamente o valor do tema no localStorage.

- Casos de teste:

1. Tema atual ‘light’. Espera-se que o tema mude para ‘dark’ e a classe ‘dark-mode’ seka adicionada.

2. Tema atual ‘dark’. Espera-se que o tema mude para ‘light’ e a classe ‘dark-mode’ seja removida.

- Complexidade Ciclomática: 2.

- Todos os testes foram executados com sucesso.

logout.test.js

- Função testada: handleLogout.

- Verifica se a função remove corretamente o token JWT do localStorage, limpa os elementos do DOM relacionados às tarefas e chama o painel de login.

- Complexidade Ciclomática: 1.

- O teste foi executado com sucesso.

**Testes de caixa preta**

loginInvalidPassword.test.js

- Verificar que, ao inserir uma senha inválida, o sistema não realiza o login e não salva nenhum token.

- Estratégia:

1. simular o preenchimento de um formulário de login HTML.

2. Preencher o campo de e-mail com ‘teste@usuario.com’.

3. Preencher o campo de senha com ‘senhaErrada’.

- O teste foi realizado com sucesso.

loginEmptyFields.test.js

- Verifica se o sistema impede o login quando os campos de email e/ou senha estão vazios, garantido a validação adequada do formulário.

- Cenário de teste:

1. Acessar a tela de login.

2. Deixar os campos de email e senha vazios.

3. Clicar no botão ‘Entrar’.

- Resultado esperado:

1. O sistema não deve permitir o login.

2. Nenhum token deve ser salvo no localStorage.

3. A tela de login deve continuar visível sem redirecionamento.

- O teste foi realizado com sucesso.

loginInvalidUser.test.js

- O teste foca na entrada (email valido) e na saída (mensagem de erro).

- Dados utilizados:

1. email: [naoexiste@teste.com](mailto:naoexiste@teste.com)

2. senha: senha123.

- Criterio de sucesso: o sistema deve exibir uma mensagem de erro como ‘Usuário não encontrado’ e não deve salvar nenhum token no localStorage.

- Resultado esperado: Mensagem de erro exibida e nenhum token salvo.

- Resultado obtido: a mensagem ‘Usuario não encontrado’ foi exibida.

- O teste foi realizado com sucesso.

cadastroSuccess.test.js

- O teste verifica se o sistema realiza corretamente o cadastro de um novo usuário quando todos os campos obrigatórios são preenchidos com dados válidos. O foco é validar a resposta externa da aplicação, sem considerar a lógica interna do código.

- Entradas:

1. Nome: joão da silva.

2. email: [joao@teste.com](mailto:joao@teste.com)

3. Senha: 123456

4. Confirmação de senha: 123456

- Comportamento esperado:

1. O sistema deve aceitar os dados fornecidos.

2. O usuário deve ser redirecionado ou receber uma confirmação de sucesso.

3. O localStorage deve armazenar o token ou os dados do usuário.

4. Nenhuma mensagem de erro deve ser exibida.

- O teste foi realizado com sucesso.

cadastroEmailExistente.test.js

- O teste verifica o comportamento do sistema quando o usuário tenta se cadastrar com o mesmo email.

- Etapas:

1. formulário de cadastro é exibido com os campos: nome, email e senha.

2. o usuário preenche o formulário com um email já existente no sistema.

3. o usuário envia o formulário clicando no botão de cadastro.

- Resultado esperado: o sistema deve impedir o cadastro, exibir uma mensagem de erro informando que o email já está em uso e redirecionar para o formulário de cadastro.

- Teste foi realizado com sucesso.

registerEmptyFields.test.js

- O teste realiza o cadastro de um novo usuário deixando todos os campos obrigatórios (nome de usuário, senha) em branco.

- Comportamento esperado: o sistema deve impedir o cadastro e exibir uma mensagem de erro ou impedir o envio do formulário.

- Caso de teste:

1. O usuário acessa o formulário de cadastro.

2. Clica no botão de enviar sem preencher nenhum campo.

3. O sistema deve acusar erro ou impedir o envio.

- Resultado esperado:

1. Cadastro não deve ser efetuado.

2. Exibição de feedback visual( mensagem de erro, destaque nos campos).

- O teste foi realizado com sucesso.

registerShortPassword.test.js

- O teste verifica se o sistema impede o cadastro quando a senha fornecida possui menos de 6 caracteres, o que viola a regra mínima de segurança.

- Entrada:

1. Nome válido.

2. Email valido.

3. Senha com menos de 6 caracteres.

- Comportamento esperado: O sistema deve exibir uma mensagem de erro e não prosseguir com o cadastro.

- O teste foi realizado com sucesso.

createTaskSuccess.test.js

- O teste verifica se o sistema permite a criação de uma tarefa quando todos os campos obrigatórios estão preenchidos corretamente.

- Entrada: usuário preenche os campos ‘titulo’, ‘descrição’ e ‘data de vencimento’ com valores válidos no formulário de nova tarefa.

-Resultado esperado: a tarefa é criada com sucesso, aparece na lista de tarefas e persiste após recarregar a página.

- Teste foi realizado com sucesso.

createTaskEmptyTitle.test.js

- O teste verifica se o sistema permite a criação de tarefa quando a descrição está vazia.

- Caso de tese:

1. acessar a interface de criação de tarefa.

2. Preencher o campo de título com um valor válido.

3. Deixar o campo de descrição e branco.

4. Preencher a data de vencimento com uma data válida.

5. Clicar em ‘criar tarefa’ no formulário.

- Resultado esperado: O sistema impede a criação de tarefa e exibe uma mensagem de erro indicando que a descrição é obrigatória.

- O teste foi realizado com sucesso.

deleteTask.test.js

- O teste verifica se a exclusão de uma tarefa no sistema resulta em sua remoção visual do DOM e se uma mensagem de sucesso é exibida para o usuário.

- Resultado esperado:

1. a tarefa com ID correspondente não deve mais existir no DOM.

2. a função showUIMessage deve ser chamada com a mensagem: ‘Tarefa removida com sucesso!’.

- O teste foi realizado com sucesso