**Prática Hands-On**

Hora de escrever seus próprios testes! Experimente estes exercícios:

**Exercício 1: Desafio de Teste Unitário**

Escreva testes unitários abrangentes para uma função de validação de senha. Teste diferentes cenários incluindo casos extremos e condições de erro:

example.js

function validarSenha(senha) {

if (!senha) return { valida: false, erro: 'Senha obrigatória' };

if (senha.length < 8) return { valida: false, erro: 'Senha muito curta' };

if (!/[A-Z]/.test(senha)) return { valida: false, erro: 'Precisa de letra maiúscula' };

if (!/[a-z]/.test(senha)) return { valida: false, erro: 'Precisa de letra minúscula' };

if (!/[0-9]/.test(senha)) return { valida: false, erro: 'Precisa de número' };

if (!/[!@#$%^&\*]/.test(senha)) return { valida: false, erro: 'Precisa de caractere especial' };

return { valida: true, mensagem: 'Senha é válida' };

}

// Seus testes unitários aqui:

function testarValidacaoSenha() {

// Teste 1: Senha vazia/nula

// Teste 2: Senha muito curta (< 8 caracteres)

// Teste 3: Faltando letra maiúscula

// Teste 4: Faltando letra minúscula

// Teste 5: Faltando número

// Teste 6: Faltando caractere especial

// Teste 7: Senha válida com todos os requisitos

// Adicione seus casos de teste abrangentes aqui!

}

**Exercício 2: Desafio de Teste de Integração**

Crie um teste de integração para um sistema de posts de blog que valida, salva e notifica assinantes:

example.js

const posts = [];

const assinantes = ['usuario1@exemplo.com', 'usuario2@exemplo.com'];

function validarPost(post) {

if (!post.titulo || post.titulo.length < 5) return { valido: false, erro: 'Título muito curto' };

if (!post.conteudo || post.conteudo.length < 50) return { valido: false, erro: 'Conteúdo muito curto' };

if (!post.autor || post.autor.length < 2) return { valido: false, erro: 'Autor obrigatório' };

return { valido: true };

}

function salvarPost(post) {

const novoPost = {

id: posts.length + 1,

...post,

criadoEm: new Date().toISOString(),

publicado: true

};

posts.push(novoPost);

return novoPost;

}

function notificarAssinantes(post) {

const notificacoes = assinantes.map(email => ({

email,

assunto: `Novo post: ${post.titulo}`,

enviado: true

}));

return `Notificados ${assinantes.length} assinantes sobre: ${post.titulo}`;

}

function publicarPost(dadosPost) {

const validacao = validarPost(dadosPost);

if (!validacao.valido) return { sucesso: false, erro: validacao.erro };

const post = salvarPost(dadosPost);

const notificacao = notificarAssinantes(post);

return { sucesso: true, post, notificacao };

}

// Seu teste de integração aqui - teste o fluxo completo!

**Exercício 3: Desafio de Teste de Interface**

Escreva um teste de interface abrangente para uma aplicação de lista de tarefas que cobre o fluxo completo do usuário:

example.js

let tarefas = [];

let proximoId = 1;

let filtro = 'todas'; // 'todas', 'ativas', 'concluidas'

function adicionarTarefa(texto) {

if (!texto || !texto.trim()) return { sucesso: false, erro: 'Não é possível adicionar tarefa vazia' };

const tarefa = { id: proximoId++, texto: texto.trim(), concluida: false, criadaEm: new Date() };

tarefas.push(tarefa);

return { sucesso: true, mensagem: `Adicionada: ${tarefa.texto}`, tarefa };

}

function alternarTarefa(id) {

const tarefa = tarefas.find(t => t.id === id);

if (!tarefa) return { sucesso: false, erro: 'Tarefa não encontrada' };

tarefa.concluida = !tarefa.concluida;

return { sucesso: true, mensagem: `${tarefa.concluida ? 'Concluída' : 'Não concluída'}: ${tarefa.texto}` };

}

function excluirTarefa(id) {

const indice = tarefas.findIndex(t => t.id === id);

if (indice === -1) return { sucesso: false, erro: 'Tarefa não encontrada' };

const excluida = tarefas.splice(indice, 1)[0];

return { sucesso: true, mensagem: `Excluída: ${excluida.texto}` };

}

function definirFiltro(novoFiltro) {

if (!['todas', 'ativas', 'concluidas'].includes(novoFiltro)) {

return { sucesso: false, erro: 'Filtro inválido' };

}

filtro = novoFiltro;

return { sucesso: true, mensagem: `Filtro definido para: ${filtro}` };

}

function obterTarefasFiltradas() {

switch (filtro) {

case 'ativas': return tarefas.filter(t => !t.concluida);

case 'concluidas': return tarefas.filter(t => t.concluida);

default: return tarefas;

}

}

// Teste o fluxo completo do usuário: adicionar, alternar, filtrar, excluir!

**💡 Dicas:**

* Use console.log() para ver resultados
* Teste cenários de sucesso e falha
* Comece simples, depois adicione mais casos de teste

// Simulação de carrinho de compras

let carrinho = [];

let estaLogado = false;

let usuarioAtual = null;

let codigoDesconto = null;

function adicionarAoCarrinho(item) {

const itemExistente = carrinho.find(itemCarrinho => itemCarrinho.id === item.id);

if (itemExistente) {

itemExistente.quantidade += item.quantidade;

} else {

carrinho.push({ ...item });

}

return `Adicionado ${item.nome} ao carrinho (Qtd: ${item.quantidade})`;

}

function atualizarQuantidade(itemId, novaQuantidade) {

const item = carrinho.find(itemCarrinho => itemCarrinho.id === itemId);

if (!item) {

return 'Item não encontrado no carrinho';

}

if (novaQuantidade <= 0) {

carrinho = carrinho.filter(itemCarrinho => itemCarrinho.id !== itemId);

return 'Item removido do carrinho';

}

item.quantidade = novaQuantidade;

return 'Quantidade atualizada com sucesso';

}

function aplicarDesconto(codigo) {

if (codigo === 'ECONOMIZE10') {

codigoDesconto = { codigo: 'ECONOMIZE10', desconto: 0.1 };

return 'Desconto aplicado: 10% de desconto';

}

return 'Código de desconto inválido';

}

function login(nomeUsuario, senha) {

if (nomeUsuario === 'usuario@exemplo.com' && senha === 'senha123') {

estaLogado = true;

usuarioAtual = { email: nomeUsuario, nome: 'João Silva' };

return 'Login realizado com sucesso';

}

return 'Credenciais inválidas';

}

function finalizarCompra() {

if (!estaLogado) {

return { sucesso: false, erro: 'Por favor, faça login primeiro' };

}

if (carrinho.length === 0) {

return { sucesso: false, erro: 'Carrinho está vazio' };

}

let total = carrinho.reduce((soma, item) => soma + (item.preco \* item.quantidade), 0);

if (codigoDesconto) {

total = total \* (1 - codigoDesconto.desconto);

}

const itensCompra = [...carrinho];

carrinho = []; // Limpar carrinho após finalização

codigoDesconto = null;

return {

sucesso: true,

idPedido: 'PED-' + Date.now(),

itens: itensCompra,

total: Math.round(total \* 100) / 100,

cliente: usuarioAtual

};

}

// Teste de Interface (Ponta a Ponta) para fluxo de compras

function testarFluxoCompras() {

// Resetar estado para teste limpo

carrinho = [];

estaLogado = false;

usuarioAtual = null;

codigoDesconto = null;

console.log('🛒 Iniciando teste de fluxo de compras...');

// Passo 1: Adicionar itens ao carrinho

const notebook = { id: 1, nome: 'Notebook', preco: 2500, quantidade: 1 };

const mouse = { id: 2, nome: 'Mouse Sem Fio', preco: 80, quantidade: 2 };

const teclado = { id: 3, nome: 'Teclado Mecânico', preco: 350, quantidade: 1 };

let resultado = adicionarAoCarrinho(notebook);

if (!resultado.includes('Adicionado Notebook ao carrinho')) {

throw new Error('Falha ao adicionar notebook ao carrinho');

}

resultado = adicionarAoCarrinho(mouse);

if (!resultado.includes('Adicionado Mouse Sem Fio ao carrinho')) {

throw new Error('Falha ao adicionar mouse ao carrinho');

}

resultado = adicionarAoCarrinho(teclado);

if (!resultado.includes('Adicionado Teclado Mecânico ao carrinho')) {

throw new Error('Falha ao adicionar teclado ao carrinho');

}

// Passo 2: Atualizar quantidades

resultado = atualizarQuantidade(2, 3); // Mudar quantidade do mouse para 3

if (resultado !== 'Quantidade atualizada com sucesso') {

throw new Error('Falha ao atualizar quantidade do mouse');

}

// Passo 3: Aplicar código de desconto

resultado = aplicarDesconto('ECONOMIZE10');

if (!resultado.includes('Desconto aplicado')) {

throw new Error('Falha ao aplicar código de desconto válido');

}

// Passo 4: Tentar código de desconto inválido

resultado = aplicarDesconto('INVALIDO');

if (!resultado.includes('Código de desconto inválido')) {

throw new Error('Código de desconto inválido deveria ser rejeitado');

}

// Passo 5: Tentar finalizar sem login (deveria falhar)

let resultadoCompra = finalizarCompra();

if (resultadoCompra.sucesso) {

throw new Error('Finalização deveria falhar sem login');

}

if (!resultadoCompra.erro.includes('Por favor, faça login primeiro')) {

throw new Error('Mensagem de erro errada para finalização sem autenticação');

}

// Passo 6: Tentar login com credenciais inválidas

resultado = login('email@errado.com', 'senhaerrada');

if (resultado !== 'Credenciais inválidas') {

throw new Error('Login inválido deveria ser rejeitado');

}

// Passo 7: Login com credenciais válidas

resultado = login('usuario@exemplo.com', 'senha123');

if (resultado !== 'Login realizado com sucesso') {

throw new Error('Login válido falhou');

}

// Passo 8: Finalizar compra

resultadoCompra = finalizarCompra();

if (!resultadoCompra.sucesso) {

throw new Error('Finalização falhou: ' + resultadoCompra.erro);

}

// Verificar detalhes do pedido

const totalEsperado = (2500 + (80 \* 3) + 350) \* 0.9; // Com 10% de desconto

if (Math.abs(resultadoCompra.total - totalEsperado) > 0.01) {

throw new Error(`Total incorreto: esperado ${totalEsperado}, obtido ${resultadoCompra.total}`);

}

if (resultadoCompra.itens.length !== 3) {

throw new Error('Número incorreto de itens no pedido');

}

if (resultadoCompra.cliente.email !== 'usuario@exemplo.com') {

throw new Error('Informações incorretas do cliente');

}

// Passo 9: Verificar se carrinho está vazio após finalização

if (carrinho.length !== 0) {

throw new Error('Carrinho deveria estar vazio após finalização');

}

// Passo 10: Tentar finalizar com carrinho vazio

resultadoCompra = finalizarCompra();

if (resultadoCompra.sucesso) {

throw new Error('Finalização deveria falhar com carrinho vazio');

}

console.log('✓ Teste de Interface (Ponta a Ponta) passou!');

console.log(`✓ Pedido ${resultadoCompra.idPedido || 'N/A'} processado com sucesso`);

}

testarFluxoCompras();

// Sistema de gerenciamento de usuários

const usuarios = [ ];

function validarUsuario(dadosUsuario) {

if (!dadosUsuario.email || !dadosUsuario.email.includes('@')) {

return { valido: false, erro: 'Formato de email inválido' };

}

if (!dadosUsuario.nome || dadosUsuario.nome.length < 2) {

return { valido: false, erro: 'Nome deve ter pelo menos 2 caracteres' };

}

if (!dadosUsuario.senha || dadosUsuario.senha.length < 6) {

return { valido: false, erro: 'Senha deve ter pelo menos 6 caracteres' };

}

return { valido: true };

}

function salvarUsuario(dadosUsuario) {

const usuario = usuarios.find(u => u.email === dadosUsuario.email);

if (usuario) {

throw new Error('Usuário já existe');

}

const novoUsuario = {

id: usuarios.length + 1,

...dadosUsuario,

criadoEm: new Date().toISOString()

};

usuarios.push(novoUsuario);

return novoUsuario;

}

function enviarEmailBoasVindas(usuario) {

// Simular envio de email com possível falha

if (usuario.email.includes('invalido')) {

throw new Error('Falha ao enviar email');

}

return `Email de boas-vindas enviado para ${usuario.email}`;

}

function registrarUsuario(dadosUsuario) {

try {

const validacao = validarUsuario(dadosUsuario);

if (!validacao.valido) {

return { sucesso: false, erro: validacao.erro };

}

const usuario = salvarUsuario(dadosUsuario);

const resultadoEmail = enviarEmailBoasVindas(usuario);

return {

sucesso: true,

usuario: usuario,

mensagem: resultadoEmail

};

} catch (error) {

return { sucesso: false, erro: error.message };

}

}

// Teste de integração para registro de usuário

function testarRegistroUsuario() {

// Limpar array de usuários para teste limpo

usuarios.length = 0;

// Teste 1: Registro bem-sucedido

const usuarioValido = {

nome: 'João Silva',

email: 'joao@exemplo.com',

senha: 'senhaSegura123'

};

const resultado = registrarUsuario(usuarioValido);

if (!resultado.sucesso) {

throw new Error('Registro de usuário válido falhou: ' + resultado.erro);

}

if (resultado.usuario.nome !== 'João Silva') {

throw new Error('Nome do usuário não foi salvo corretamente');

}

if (!resultado.mensagem.includes('Email de boas-vindas enviado')) {

throw new Error('Email de boas-vindas não foi enviado');

}

if (usuarios.length !== 1) {

throw new Error('Usuário não foi salvo no banco de dados');

}

// Teste 2: Registro de usuário duplicado

const resultadoDuplicado = registrarUsuario(usuarioValido);

if (resultadoDuplicado.sucesso) {

throw new Error('Usuário duplicado não deveria se registrar');

}

if (usuarios.length !== 1) {

throw new Error('Usuário duplicado foi salvo no banco de dados');

}

// Teste 3: Registro com email inválido

const usuarioInvalido = {

nome: 'Maria Silva',

email: 'email-invalido',

senha: 'senha123'

};

const resultadoInvalido = registrarUsuario(usuarioInvalido);

if (resultadoInvalido.sucesso) {

throw new Error('Usuário inválido não deveria se registrar');

}

if (!resultadoInvalido.erro.includes('Formato de email inválido')) {

throw new Error('Mensagem de erro errada para email inválido');

}

// Teste 4: Falha no envio de email

const usuarioFalhaEmail = {

nome: 'Usuário Teste',

email: 'invalido@exemplo.com',

senha: 'senha123'

};

const resultadoFalhaEmail = registrarUsuario(usuarioFalhaEmail);

if (resultadoFalhaEmail.sucesso) {

throw new Error('Registro deveria falhar quando envio de email falha');

}

console.log('✓ Teste de integração passou!');

}

testarRegistroUsuario();