**TICKET SUPORTE**

Ticket 1: Falha ao Processar Pagamento

• Descrição: Os clientes estão relatando que, ao tentar realizar pagamentos, o sistema trava e retorna uma mensagem de erro: "Pagamento não processado. Tente novamente mais tarde".

• Possível Causa: A classe CarrinhoDeCompras está diretamente acoplada à classe Pagamento, resultando em falhas quando o método de pagamento é atualizado sem que o carrinho seja atualizado.

• Tarefa: Identificar o impacto do alto acoplamento entre as classes e sugerir como isso poderia ser solucionado (como por meio de interfaces de pagamento).

RESPOSTA-1: Os principais impactos do alto acoplamento seria a Dificuldade de manutenção, Baixa flexibilidade, Teste complicado e a Fragilidade do sistema.  
Uma sugestão para solucionar este problema seria a implementação de uma interface de pagamento que declare os métodos necessários para processar pagamentos, implementar classes de pagamento e por ultimo modificar a classe CarrinhoDeCompras para depender da Interface

Ticket 2: Falha Intermitente na Autenticação

• Descrição: Alguns usuários estão enfrentando problemas intermitentes ao tentar fazer login. Eles relatam que, após inserir as credenciais, o sistema simplesmente recarrega a página.

• Possível Causa: O sistema de autenticação está diretamente ligado ao serviço de banco de dados sem tratamento de erro apropriado, o que pode resultar em falhas quando o serviço de autenticação fica indisponível.

• Tarefa: Identificar possíveis pontos de falha no processo de login e sugerir melhorias no manuseio de erros e no desacoplamento do serviço de autenticação.

RESPOSTA-2: Os possíveis pontos de falha no processo de login estão ligados a dependência direta do banco de dados, a falta de tratamento de erros, validação inadequada das credenciais e o feedback inadequado ao usuário.  
Uma sugestão é a implementação de um padrão de tratamento de erros junto com um desacoplamento do serviço de autenticação e um sistema de retentativas, possibilitando que o usuário possa tentar se conectar um número definido de vezes antes de falhar.

**TICKET DESENVOLVEDORES**

Ticket 1: Falha ao Processar Pagamento

class IPagamento:

def efetuar\_pagamento(self, valor):

pass

class CarrinhoDeCompras:

def \_\_init\_\_(self, pagamento: IPagamento):

self.pagamento = pagamento

self.\_itens = []

def adicionar\_item(self, valor):

self.\_itens.append(valor)

def total(self):

return sum(self.\_itens)

def processar\_pagamento(self):

self.pagamento.efetuar\_pagamento(self.total())

class PagamentoCartao(IPagamento):

def efetuar\_pagamento(self, valor):

print(f"Pagamento de R${valor:.2f} efetuado com cartão.")

carrinho = CarrinhoDeCompras(PagamentoCartao())

carrinho.adicionar\_item(50.0)

carrinho.adicionar\_item(75.0)

carrinho.processar\_pagamento()

Ticket 2: Falha Intermitente na Autenticação

class ServicoBancoDeDados:

def verificar\_usuario(self, usuario, senha):

return usuario == "admin" and senha == "1234"

class Autenticacao:

def \_\_init\_\_(self, banco: ServicoBancoDeDados):

self.banco = banco

def autenticar(self, usuario, senha):

if self.banco.verificar\_usuario(usuario, senha):

print("Usuário autenticado com sucesso!")

else:

print("Falha na autenticação. Verifique suas credenciais.")

banco = ServicoBancoDeDados()

servico\_autenticacao = Autenticacao(banco)

servico\_autenticacao.autenticar("admin", "1234") # Usuário autenticado com sucesso!

servico\_autenticacao.autenticar("user", "wrong") # Falha na autenticação. Verifique suas credenciais.

Ticket 3: Relatórios Demoram para Carregar

class ServicoBancoDeDados:

def consultar\_dados\_essenciais(self):

# Simula uma consulta otimizada

return ["Dado 1", "Dado 2", "Dado 3"]

def consultar\_detalhes(self, dado):

return f"Detalhes do {dado}"

class Relatorio:

def \_\_init\_\_(self, banco: ServicoBancoDeDados):

self.banco = banco

def gerar(self):

for dado in self.banco.consultar\_dados\_essenciais():

print(dado)

print(self.banco.consultar\_detalhes(dado))

banco = ServicoBancoDeDados()

relatorio = Relatorio(banco)

relatorio.gerar()

Ticket 4: Erro ao Atualizar Perfil do Usuário

class Endereco:

def atualizar(self, novo\_endereco):

print(f"Endereço atualizado para: {novo\_endereco}")

class Perfil:

def \_\_init\_\_(self, endereco: Endereco):

self.endereco = endereco

def atualizar\_endereco(self, novo\_endereco):

self.endereco.atualizar(novo\_endereco)

endereco = Endereco()

perfil = Perfil(endereco)

perfil.atualizar\_endereco("Rua Nova, 123")

Ticket 5: Logout Automático Inesperado

class GerenciadorDeSessao:

def \_\_init\_\_(self):

self.sessao\_ativa = True

def monitorar\_sessao(self):

if self.inativo():

self.logout()

def inativo(self):

return False

def logout(self):

self.sessao\_ativa = False

print("Usuário desconectado.")

gerenciador\_sessao = GerenciadorDeSessao()

gerenciador\_sessao.monitorar\_sessao()

Ticket 6: Falha no Carregamento de Imagens no Front-end

class CarregadorImagens:

def \_\_init\_\_(self, backend):

self.backend = backend

def carregar(self):

return self.backend.recuperar\_imagens()

class Backend:

def recuperar\_imagens(self):

return ["imagem1.png", "imagem2.jpg", "imagem3.gif"]

backend = Backend()

carregador = CarregadorImagens(backend)

imagens = carregador.carregar()

for imagem in imagens:

print(f"Carregou: {imagem}")

Ticket 7: Falha ao Adicionar Produto no Carrinho

class Carrinho:

def \_\_init\_\_(self):

self.produtos = []

def adicionar\_produto(self, produto):

self.produtos.append(produto)

return f"Produto '{produto}' adicionado ao carrinho."

class ApiCarrinho:

def \_\_init\_\_(self, carrinho: Carrinho):

self.carrinho = carrinho

def adicionar\_produto(self, produto):

resultado = self.carrinho.adicionar\_produto(produto)

return resultado

carrinho = Carrinho()

api\_carrinho = ApiCarrinho(carrinho)

resultado = api\_carrinho.adicionar\_produto("Produto A")

print(resultado)

print(f"Produtos no carrinho: {carrinho.produtos}")

Ticket 8: Falha no Envio de Notificações

class ServicoEmail:

def enviar(self, usuario):

print(f"E-mail enviado para: {usuario}")

class Sistema:

def \_\_init\_\_(self, servico\_email: ServicoEmail):

self.servico\_email = servico\_email

def notificar\_usuario(self, usuario):

self.servico\_email.enviar(usuario)

servico\_email = ServicoEmail()

sistema = Sistema(servico\_email)

sistema.notificar\_usuario("usuario@exemplo.com")

Ticket 9: Excesso de Dados ao Consultar Histórico de Compras

class BancoDeDados:

def consultar\_compras(self, usuario, limite=10, offset=0):

todas\_as\_compras = [

"Compra 1", "Compra 2", "Compra 3",

"Compra 4", "Compra 5", "Compra 6",

"Compra 7", "Compra 8", "Compra 9", "Compra 10",

"Compra 11", "Compra 12"

]

return todas\_as\_compras[offset:offset + limite]

class HistoricoCompras:

def \_\_init\_\_(self, banco\_de\_dados):

self.banco\_de\_dados = banco\_de\_dados

def carregar\_historico(self, usuario, pagina=1):

limite = 5 # Limite fixo de compras por página

offset = (pagina - 1) \* limite

return self.banco\_de\_dados.consultar\_compras(usuario, limite, offset)

banco\_de\_dados = BancoDeDados()

historico = HistoricoCompras(banco\_de\_dados)

for pagina in range(1, 4): # Carrega 3 páginas

compras = historico.carregar\_historico("usuario1", pagina)

print(f"Compras na página {pagina}:", compras)

Ticket 10: Falha ao Alterar Status de Pedido

class Pedido:

def \_\_init\_\_(self, cliente):

self.cliente = cliente

self.status = "Pendente"

def alterar\_status(self, novo\_status):

servico\_status = ServicoStatus()

servico\_status.alterar(self, novo\_status)

class ServicoStatus:

def alterar(self, pedido, novo\_status):

pedido.status = novo\_status

print(f"Status do pedido alterado para: {novo\_status}")

class Cliente:

def \_\_init\_\_(self, nome):

self.nome = nome

cliente = Cliente("João")

pedido = Pedido(cliente)

pedido.alterar\_status("Enviado")