Plano de Teste para o Sistema de Reservas de Quartos de Hotel

Versão 1.0

Histórico da Revisão

Data	Versão	Descrição	Autor
9/11/2023	1.0	Release Inicial - Protótipo do Plano de Teste	Lucas Maciel

Ìndice

Plano de Teste
para o
Sistema de Reservas de Quartos de Hotel

1. Objetivos

1.1 Objetivo

Este documento descreve o plano para testar o sistema de reservas de quartos de hotel. Este documento de Plano de Teste suporta os seguintes objetivos:

- Identificar informações existentes do projeto e o software que deve ser testado.
- Listar os requisitos de teste recomendados (nível alto).
- Recomendar e descrever as estratégias de teste a serem empregadas.
- Fornecer uma estimativa dos esforços de teste.

1.2 Escopo

As seguintes interfaces serão testadas:

- Cadastros
- Gestão e consultas
- Pagamentos
- Relatórios

2. Requisitos para o Teste

A lista a seguir identifica os itens (casos de uso, requisitos funcionais) que foram identificados como alvos do teste. Essa lista representa o que será testado.

- Cadastro de Quartos
- Cadastro de Hóspedes
- Consulta de Disponibilidade
- Reserva de Quartos
- Cancelamento de Reservas
- Pagamento
- Gestão de Estadia
- Relatórios

3. Estratégia do Teste

A Estratégia de Teste apresenta a abordagem recomendada para o teste dos aplicativos de software. A seção anterior dos Requisitos de Teste descrevia o que será testado; esta descreve como será testado.

As principais considerações para a estratégia de teste são as técnicas a serem utilizadas e o critério para saber quando o teste está concluído.

Além das considerações fornecidas para cada teste a seguir, o teste deve ser executado apenas utilizando bancos de dados conhecidos e controlados, em ambientes protegidos.

A estratégia de teste a seguir é genérica por natureza e foi desenvolvida para ser aplicada aos requisitos listados na seção 4 deste documento.

3.1 Tipos de Teste

3.1.1. Cadastro de Quartos

Objetivo: Verificar se é possível cadastrar quartos corretamente no sistema.

Passos:
Acesse a funcionalidade de cadastro de quartos.
Preencha os campos obrigatórios (número do quarto, tipo, capacidade, preço).
Clique em "Salvar".

Verifique se o quarto foi cadastrado corretamente.

Critérios de Aceitação:
O quarto deve ser salvo com sucesso no banco de dados.
Deve ser possível visualizar o quarto cadastrado na lista de quartos.

3.1.2 Cadastro de Hóspedes

Objetivo: Garantir que os hóspedes possam ser cadastrados no sistema.

Passos:
Acesse a funcionalidade de cadastro de hóspedes.
Preencha os campos obrigatórios (nome, sobrenome, data de nascimento, cpf, gênero, endereço, telefone, e-mail).
Clique em "Salvar".
Verifique se o hóspede foi cadastrado corretamente.

Critérios de Aceitação:
O hóspede deve ser salvo com sucesso no banco de dados.
Deve ser possível visualizar o hóspede cadastrado na lista de hóspedes.

3.1.3 Consulta de Disponibilidade

Objetivo: Testar se é possível consultar a disponibilidade de quartos para determinadas datas.

Passos:
Acesse a funcionalidade de consulta de disponibilidade.
Escolha as datas de check-in e check-out.
Clique em "Consultar Disponibilidade".
Verifique se a lista de quartos disponíveis é exibida corretamente.

Critérios de Aceitação:
A lista de quartos disponíveis deve ser precisa para as datas especificadas.

3.1.4 Reserva de Quartos

Objetivo: Testar se é possível fazer uma reserva de quarto com sucesso.

Passos:
Selecione um quarto disponível.
Escolha as datas de check-in e check-out.
Preencha os dados do hóspede.
Clique em "Reservar Quarto".
Verifique se a reserva é confirmada e o quarto é marcado como indisponível.

Critérios de Aceitação:

A reserva deve ser registrada com sucesso no banco de dados.
O quarto reservado deve ser marcado como indisponível para as datas selecionadas.

3.1.5 Cancelamento de Reservas

Objetivo: Testar se é possível cancelar uma reserva existente.

Passos:
Acesse a funcionalidade de cancelamento de reservas.
Selecione a reserva que deseja cancelar.
Confirme o cancelamento.
Verifique se a reserva é cancelada e o quarto volta a ficar disponível.

Critérios de Aceitação:
A reserva deve ser cancelada com sucesso no banco de dados.
O quarto reservado deve voltar a ficar disponível.

3.1.6 Pagamento

Objetivo: Verificar se o sistema processa o pagamento corretamente.

Passos:
Acesse a funcionalidade de pagamento.
Selecione a reserva que deseja pagar.
Insira os detalhes do pagamento (número do cartão, data de validade, código de segurança).
Confirme o pagamento.
Verifique se o pagamento é processado com sucesso e a reserva é marcada como paga.

Critérios de Aceitação:
O pagamento deve ser processado corretamente.
A reserva deve ser marcada como paga no banco de dados.

3.1.7 Gestão de Estadia
Objetivo: Testar se é possível gerenciar a estadia de um hóspede (check-in, check-out).

Passos:
Acesse a funcionalidade de gestão de estadia.
Selecione um hóspede que fez uma reserva.
Realize o check-in do hóspede.
Realize o check-out do hóspede.
Verifique se a estadia é gerenciada corretamente.

Critérios de Aceitação:
O check-in e check-out devem ser registrados corretamente no banco de dados.
O quarto reservado deve ficar disponível após o check-out.

3.1.8 Relatórios

Objetivo: Verificar se os relatórios são gerados corretamente.

Passos:
Acesse a funcionalidade de geração de relatórios.
Selecione o tipo de relatório desejado (por exemplo, ocupação do hotel, receita).
Escolha o período de tempo para o relatório.
Clique em "Gerar Relatório".
Verifique se o relatório é exibido corretamente.

Critérios de Aceitação:
O relatório deve conter as informações corretas de acordo com o tipo e período selecionados.

4.Marcos do Projeto

O teste do Sistema de Reservas de Quartos de Hotel incorpora tarefas de teste para cada um dos esforços de teste identificados nas seções anteriores. Marcos do projeto separados são identificados para comunicar o status e as realizações do projeto.

Tarefa de Marco	Esforço (pd)	Data de Início	Data de Encerramento
Planejamento de Teste do sistema	2	02 de novembro	09 de novembro
Design de Teste do sistema	3	09 de novembro	15 de novembro
Desenvolvimento de Teste do sistema	4	16 de novembro	23 de novembro
Execução de Teste do sistema	3	24 de novembro	01 de dezembro
Avaliação de Teste do Protótipo	1	02 de dezembro	09 de dezembro

5.Tarefas do Projeto

A seguir são mostradas as tarefas relacionadas ao teste do Sistema de Reservas de Quartos de Hotel:

Planejar Teste

Identificar Requisitos para o Teste

Avaliar Risco

Desenvolver Estratégia de Teste

Identificar Recursos de Teste

Criar Planejamento

Gerar Plano de Teste

Projetar Teste

Desenvolver Conjunto de Teste

Identificar e Descrever Casos de Teste

Identificar e Estruturar Scripts de Teste

Revisar e Acessar Cobertura de Teste

Implementar Teste

Configurar Ambiente de Teste

Registrar ou Programar Scripts de Teste

Desenvolver Drivers e Stubs de Teste

Identificar funcionalidade específica do Teste no design e modelo de implementação

Estabelecer Conjuntos de Dados Externos

Executar Teste

Executar Script de Teste

Avaliar Execução do Teste

Recuperar-se de Teste Interrompido

Verificar os Resultados

Investigar Resultados Inesperados

Registrar Defeitos

Avaliar Teste

Avaliar a Cobertura dos Casos de Teste

Avaliar Cobertura do Código

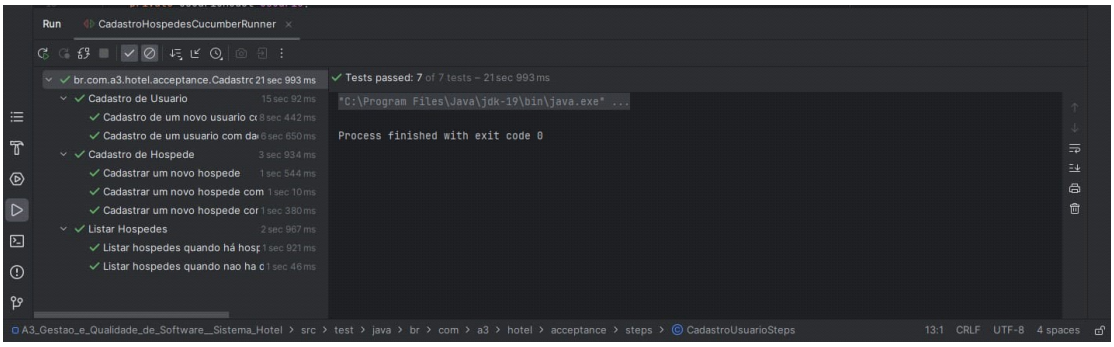
Analisar Defeitos

Determinar se os Critérios de Conclusão do Teste e os Critérios de Êxito foram alcançados

Criar Relatório de Avaliação do Teste

6 - Print do Resultado

Os cenários foram executados todos de uma só vez, não apresentando falhas.



7 - Linguagem Gherkin:

<u>CadastroHospedes</u>

Cadastro com sucesso:
Dado que o usuário está na tela de cadastro de hóspede,
Quando o usuário insere os dados
Entao o sistema deve cadastrar o hóspede no banco de dados.
Cadastro sem sucesso:

Dado que o usuário está na tela de cadastro de hospede,
Quando o usuário insere os dados
Entao o sistema não deve cadastrar o hospede no banco de dados.

<u>CadastroUsuario</u>

Cadastro com sucesso
Dado que o usuário está na tela de cadastro
Quando o usuário acessa a opção de cadastrar usuário
Então o usuário deve ser criado com sucesso
Cadastro sem sucesso:
Dado que o usuário está na tela de cadastro
Quando o usuário acessa a opção de cadastrar usuário
Então o usuário não deve ser criado.

ListarHospedes

Listagem informa os dados
Dado que existem hóspedes cadastrados no sistema,
Quando o usuário acessa a opção de listar hospedes
Então uma lista de hóspedes deve ser retornada.
Listagem não informa os dados
Dado que não existem hóspedes cadastrados no sistema,
Quando o usuário acessa a opção de listar hospedes
Então uma lista vazia deve ser retornada.

8 - Códigos aplicados nos cenários:

```
public class CadastroHospedesSteps {

    private HospedesModel hospede;
    private HospedesView hospedesView;
    private DAO.HospedesDAO hospedesDAO;

    @Dado("que o usuario está na tela de cadastro de hospede")
    public void que_o_usuario_está_na_tela_de_cadastro_de_hospede() {
        hospedesView = new HospedesView();
    }
    @Quando("o usuario insere os seguintes dados:")
    public void o_usuario_insere_os_seguintes_dados(DataTable dataTable) {

        List<Map<String, String>> valores = dataTable.asMaps();

        String nome = null;
        String sobrenome = null;
        String dtNascimento = null;
        String cpf = null;
        String genero = null;
        String endereco = null;
        String telefone = null;
        String email = null;

        for (Map<String, String> mapa : valores){

            nome = mapa.get("nome");
            sobrenome = mapa.get("sobrenome");
            dtNascimento = mapa.get("dtNascimento");
            cpf = mapa.get("cpf");
            genero = mapa.get("genero");
            endereco = mapa.get("endereco");
            telefone = mapa.get("telefone");
            email = mapa.get("email");
        }

        hospede = new HospedesModel(nome, sobrenome, dtNascimento, cpf, genero, endereco,
telefone, email);
    }
```

```
    @Entao("o sistema deve cadastrar o hospede no banco de dados")
    public void o_sistema_deve_cadastrar_o_hospede_no_banco_de_dados() throws SQLException,
ClassNotFoundException {
        hospedesDAO = new HospedesDAO();
        boolean cadastroHospede = hospedesDAO.cadastrarHospede(hospede);

        assertEquals(true, cadastroHospede);
    }

    @Entao("o sistema não deve cadastrar o hospede no banco de dados")
    public void o_sistema_não_deve_cadastrar_o_hospede_no_banco_de_dados() throws
SQLException, ClassNotFoundException {
        hospedesDAO = new HospedesDAO();
        boolean cadastroHospede = hospedesDAO.cadastrarHospede(hospede);

        assertEquals(false, cadastroHospede);
    }
}
```

```
public class CadastroUsuarioSteps {

    private UsuarioModel usuario;
    private UsuarioDAO usuarioDAO;

    @Dado("que foram preenchidos os seguintes dados:")
    public void que_foram_preenchidos_os_seguintes_dados(DataTable dataTable) {

        List<Map<String, String>> valores = dataTable.asMaps();

        String nomeUsuario = null;
        String funcional = null;
        String senha = null;

        for (Map<String, String> mapa : valores){
            nomeUsuario = mapa.get("NomeUsuario");
            funcional = mapa.get("Funcional");
            senha = mapa.get("Senha");
        }
        usuario = new UsuarioModel(nomeUsuario, funcional, senha);

    }
    @Quando("o usuario acessa a opcao de cadastrar usuario")
    public void o_usuario_acessa_a_opcao_de_cadastrar_usuario() {
        usuarioDAO = new UsuarioDAO();
    }
    @Entao("o usuario deve ser criado com sucesso")
    public void o_usuario_deve_ser_criado_com_sucesso() throws SQLException,
ClassNotFoundException {
        boolean cadastroUsuario = usuarioDAO.cadastrarUsuario(usuario);
```

```
        assertEquals(true, cadastroUsuario);
    }

    @Entao("o usuario nao deve ser criado")
    public void o_usuario_nao_deve_ser_criado() throws SQLException, ClassNotFoundException {
        boolean cadastroUsuario = usuarioDAO.cadastrarUsuario(usuario);

        assertEquals(false, cadastroUsuario);
    }
}
```

```
public class ListarHospedesSteps {

    private DAO.HospedesDAO hospedesDAO;
    private List<HospedesModel> listaDeHospedesRetornada;
    static Connection conn;

    @Dado("que existem hospedes cadastrados no sistema")
    public void que_existem_hospedes_cadastrados_no_sistema() throws SQLException,
ClassNotFoundException {
        hospedesDAO = new HospedesDAO();
        hospedesDAO.cadastrarHospede(new HospedesModel("João", "da Silva", "01/01/1980",
"12345678900", "m", "Rua Teste", "31999999999", "joao@email"));
    }
    @Quando("o usuario acessa a opção de listar hospedes")
    public void o_usuario_acessa_a_opção_de_listar_hospedes() throws SQLException,
ClassNotFoundException {
        listaDeHospedesRetornada = hospedesDAO.listarHospedes();
    }
    @Entao("uma lista de hospedes deve ser retornada")
    public void uma_lista_de_hospedes_deve_ser_retornada() {
        assertTrue(listaDeHospedesRetornada.size() > 0);
    }

    @Dado("que nao existem hospedes cadastrados no sistema")
    public void que_nao_existem_hospedes_cadastrados_no_sistema() throws SQLException,
ClassNotFoundException {
        new ConexaoDAO();
        conn = ConexaoDAO.conectaBD();

        String sql = "DELETE FROM Hospedes;";
        PreparedStatement pstm = conn.prepareStatement(sql);
        pstm.executeUpdate();
        pstm.close();
    }
    @Quando("o usuario acessa a opcao de listar hospedes")
```

```
    public void o_usuario_acessa_a_opcao_de_listar_hospedes() throws SQLException,
ClassNotFoundException {
        listaDeHospedesRetornada = hospedesDAO.listarHospedes();
    }
    @Entao("uma lista vazia deve ser retornada")
    public void uma_lista_vazia_deve_ser_retornada() {
        assertTrue(listaDeHospedesRetornada.isEmpty());
    }
}
```