RELATÓRIO SISTEMA DE EVENTOS VIP

Alunos: Gabriel Marques, Eduardo Castelo Branco, Vinicius Gomes, Clidenor De Carvalho

1. Introdução ao Projeto

O projeto **Eventos VIP** consiste em um sistema de gerenciamento para uma empresa de organização de eventos. O objetivo principal é controlar o fluxo de convidados, mesas, garçons e pedidos, com uma distinção clara entre convidados da categoria "Regular" e "VIP".

O sistema foi desenvolvido em **Java**, utilizando a arquitetura **MVC (Model-View-Controller)** para garantir a organização e a manutenibilidade do código. A solução foca na aplicação prática dos pilares da Programação Orientada a Objetos (POO), garantindo robustez no tratamento de dados e segurança nas regras de negócio.

2. Funcionalidades Implementadas

O grupo implementou todas as funcionalidades obrigatórias descritas no enunciado do problema, além de requisitos diferenciais:

1. **Gestão de Convidados:** Cadastro de convidados com diferenciação via herança (`ConvidadoRegular` e `ConvidadoVIP`).
2. **Controle de Mesas:** Alocação de convidados em mesas com validação de capacidade máxima (limite de 8 pessoas) e associação de garçons.
3. **Sistema de Pedidos:** Cardápio com itens variados. Implementação de regra de negócio que impede convidados Regulares de pedirem itens exclusivos VIP (ex: "Lagosta").
4. **Faturamento e Polimorfismo:** Cálculo automático da conta da mesa. O sistema aplica descontos automaticamente dependendo do tipo do convidado (VIPs recebem 15% de desconto), utilizando sobrescrita de métodos (`@Override`).
5. **Persistência de Dados:** O sistema salva a lista de convidados em arquivo físico (`convidados.csv`), permitindo que os dados sejam recuperados após fechar o programa.
6. **Geração de Relatórios:**
    - Relatório em Console: Resumo financeiro detalhado.
    - Relatório em PDF: Geração de arquivo PDF formatado utilizando a biblioteca externa **iText 7**.

3. Funcionalidades Não Implementadas

Todas as funcionalidades requisitadas no documento *S3a - Eventos VIP.pdf* foram implementadas.

- **Observação sobre Decoração:** O requisito de "decoração personalizada para mesas VIP" foi abstraído como uma característica informativa no momento do cadastro da mesa/evento, focando o desenvolvimento na lógica complexa de faturamento e persistência, que eram os pontos críticos da avaliação.

4. Experiência do Aluno

4.1 O que mais gostou

O ponto alto do desenvolvimento foi a implementação da **Persistência de Dados** e a integração com o **Google Drive/OneDrive**. Ver o sistema gerando um arquivo PDF e ele aparecendo automaticamente na pasta sincronizada da nuvem trouxe uma percepção real de como sistemas corporativos funcionam.

4.2 Maiores Dificuldades

A maior dificuldade foi o gerenciamento de dependências externas. Inicialmente, houve problemas para configurar as bibliotecas do **iText** manualmente (baixando arquivos `.jar`). A solução foi migrar o projeto para **Maven**, o que automatizou o download das dependências (`kernel`, `layout`, `io`, `slf4j`) e resolveu os conflitos de *classpath*.

4.3 Quanto Aprendeu

O projeto consolidou o conhecimento sobre **Tratamento de Erros**. Aprender a usar `try-catch` para blindar o sistema contra entradas inválidas (ex: letras em campos numéricos) e criar exceções personalizadas (`MesaLotadaException`, `PermissaoException`) foi essencial para elevar a qualidade do código.

5. Bibliotecas Utilizadas

Para atender aos requisitos de relatórios avançados, foram utilizadas as seguintes bibliotecas de terceiros:

1. **iText 7 Core (com.itextpdf):**
    - *Motivo:* Escolhida por ser a biblioteca padrão de mercado para criação de PDFs em Java. Ela permite criar documentos estruturados (parágrafos, alinhamentos) via código.
    - *Módulos:* `kernel`, `io`, `layout`, `forms`.
2. **SLF4J (Simple Logging Facade for Java):**
    - *Motivo:* Dependência obrigatória do iText para gerenciamento de logs. Sem ela, a geração do PDF falhava.

6. Diferenciais do Projeto

Além do solicitado, o projeto conta com:

1. **Integração com Nuvem (Desafio):** O sistema salva o relatório PDF diretamente na pasta local do **Google Drive/OneDrive**, realizando o backup automático na nuvem.
2. **Validações Robustas:** Uso de **Regex** para impedir números no nome do convidado e validações lógicas para impedir valores negativos em mesas ou IDs.

3. **Arquitetura Maven:** Migração completa para Maven, facilitando a execução do projeto em qualquer máquina sem configuração manual de bibliotecas.

7. Trecho de Código Importante

Abaixo, destaca-se a classe **`RelatorioPDFService`**, pois ela integra a biblioteca externa e demonstra a capacidade do sistema de exportar dados.

RelatorioPDFService.java

```java
package service;
import model.Convidado;
import model.Evento;
import model.Mesa;
import com.itextpdf.kernel.pdf.PdfDocument;
import com.itextpdf.kernel.pdf.PdfWriter;
import com.itextpdf.layout.Document;
import com.itextpdf.layout.element.Paragraph;
import com.itextpdf.layout.properties.TextAlignment;

import java.io.File;
import java.io.FileNotFoundException;

public class RelatorioPDFService {  2 usages   gabbms
    public void gerarRelatorioPdf(Evento evento, String caminhoArquivo) {  1 usage   gabbms
        // Cria diretório se não existir
        File arquivo = new File(caminhoArquivo);
        if (arquivo.getParentFile() != null) {
            arquivo.getParentFile().mkdirs();
        }

        try {
            PdfWriter writer = new PdfWriter(caminhoArquivo);
            PdfDocument pdf = new PdfDocument(writer);
            Document document = new Document(pdf);

            // Título
            document.add(new Paragraph( text: "Relatório Final: " + evento.getTema())
                    .setTextAlignment(TextAlignment.CENTER)
                    .setFontSize(20)
                    .setBold());

            document.add(new Paragraph( text: "\n-------------------------------------------------\n"));

            // Dados
            PagamentoService pagamentoService = new PagamentoService();
            double totalGeral = 0;

            for (Mesa mesa : evento.getMesas()) {
                document.add(new Paragraph( text: "Mesa " + mesa.getNumero()).setBold().setFontSize(14));

                double totalMesa = pagamentoService.calcularContaMesa(mesa);
                totalGeral += totalMesa;

                document.add(new Paragraph( text: "Garçom: " + mesa.getGarcomAssociado().getNome()));
                document.add(new Paragraph( text: "Convidados: " + mesa.getConvidados().size()));

                for(Convidado c : mesa.getConvidados()){
                    document.add(new Paragraph( text: " - " + c.getNome() + " (" + c.getTipo() + ")"));
                }

                document.add(new Paragraph(String.format("Total da Mesa: R$ %.2f", totalMesa)));
                document.add(new Paragraph( text: "\n"));
            }

            document.add(new Paragraph( text: "-------------------------------------------------"));
            document.add(new Paragraph(String.format("FATURAMENTO TOTAL: R$ %.2f", totalGeral))
                    .setBold().setFontSize(16));

            document.close();
            System.out.println("PDF gerado com sucesso em: " + caminhoArquivo);

        } catch (FileNotFoundException e) {
            System.err.println("Erro: Arquivo aberto ou caminho inválido.");
        } catch (Exception e) {
            System.err.println("Erro ao gerar PDF: " + e.getMessage());
        }
    }
}
```

8. Referências Consultadas

1. **MvnRepository:** Para localizar as dependências corretas do iText 7 e SLF4J.
2. **Documentação iText:** Tutoriais sobre como criar parágrafos e documentos `PdfWriter`.