Sistema de Gerenciamento de Eventos

Lucas da Conceição Damasceno

Engenharia de Computação
Universidade Estadual de Feira de Santana (UEFS) – Feira de Santana, BA – Brasil
Avenida Transnordestina s/n, Novo Horizonte – Feira de Santana – Ba – Brasil

lucas.damasceno.dev@gmail.com

Resumo. *Este relatório aborda o desenvolvimento de um sistema de gerenciamento de vendas de ingressos, projetado para simplificar o processo de compra e administração de eventos culturais, como shows, peças de teatro e sessões de cinema. Com funcionalidades para cadastro de eventos, controle de assentos, aquisição e cancelamento de ingressos, o sistema foi estruturado em uma arquitetura modular utilizando Java e aplicando testes automatizados com JUnit. A implementação enfatiza segurança e confiabilidade, com controle de permissões e tratamento de exceções em múltiplos níveis, criando uma base sólida para expansões futuras, como integração com plataformas de pagamento e aprimoramento das funcionalidades de segurança.*

1. Introdução

A fim de modernizar o sistema de vendas de ingressos, uma empresa de entretenimento decidiu adotar uma nova solução que permita a gestão eficiente e segura de eventos, como shows, peças de teatro e sessões de cinema. O desenvolvimento deste sistema buscou atender a múltiplas necessidades organizacionais e dos usuários, com foco na flexibilidade, modularidade e persistência dos dados. Para isso, implementamos funcionalidades que englobam o cadastro de eventos, gerenciamento de assentos, compra e cancelamento de ingressos, e interações adicionais, como a edição de perfil e a adição de diferentes formas de pagamento.

A solução proposta foi desenvolvida em uma estrutura de classes modular, que permite uma organização clara e bem documentada do código. Cada classe e método possui visibilidade apropriada, e foram incluídos getters e setters para acesso controlado aos atributos, conforme necessário. Para facilitar o entendimento e manutenção, o código foi comentado de maneira estruturada, o que também simplifica a realização de testes, permitindo uma validação consistente do comportamento das funcionalidades.

A persistência dos dados é garantida por um sistema que lê e grava arquivos JSON, assegurando que as informações sejam armazenadas e recuperadas corretamente. Funções essenciais, como cadastro e atualização de eventos e de usuários, além da possibilidade de associar diferentes métodos de pagamento, garantem uma experiência completa e personalizada ao usuário. Outras funcionalidades como confirmação de compra via email e avaliação de eventos foram implementadas, proporcionando um ciclo completo de atendimento e interação com o sistema.

Esta introdução fornece um panorama geral dos objetivos e do desenvolvimento do sistema até seu estado atual. Os tópicos a seguir abordarão detalhadamente a organização das classes, os métodos implementados, as estratégias de persistência de dados e os

resultados obtidos no uso das funcionalidades, discutindo a adequação das soluções aplicadas para alcançar uma experiência de venda de ingressos segura e eficiente.

2. Resultados e Discussões

A implementação do sistema de gerenciamento de eventos revelou uma série de desafios e oportunidades. A modularização do sistema, a escolha das tecnologias e a estratégia de tratamento de erros foram cruciais para o sucesso do projeto. Nesta seção, serão apresentados os resultados obtidos, discutindo os aspectos positivos e negativos da implementação, bem como as lições aprendidas durante o desenvolvimento.

2.1. Bugs e Modularização

A modularização foi crucial para o desenvolvimento e manutenção do sistema, facilitando a identificação e correção de bugs ao isolar cada funcionalidade em classes e métodos específicos. A divisão em módulos distintos, como UserController, EventController, TicketController e PurchaseController, permitiu o isolamento de responsabilidades e simplificou a manutenção. No entanto, durante a integração entre esses módulos, surgiram alguns desafios, especialmente ao lidar com as dependências entre os controladores, o que exigiu um mapeamento cuidadoso das exceções e dependências.

Um exemplo significativo é o método purchaseTicket, cuja responsabilidade inclui o processamento da compra de ingressos. Inicialmente, essa função apresentou problemas de duplicação de reserva, lançando exceções como TicketAlreadyExistsException e PurchaseException. A dependência entre PurchaseController e TicketController exigiu uma estratégia de validação pré-execução, o que minimizou o número de erros gerados. Para isso, foram adicionadas verificações de estado e métodos de controle de execução nos controladores e na camada `DataAccessObject` (DAO), promovendo maior consistência e robustez no sistema como um todo.

2.2. Diagramas de Classes e de Casos de Uso

Para detalhar a organização e o fluxo do sistema, foram elaborados diagramas de classes e casos de uso, os quais evidenciam a estrutura de interações entre usuários e administradores. No diagrama de classes, observa-se que User e Event são as principais classes de dados, gerenciando as permissões dos usuários e as informações dos eventos, respectivamente. Controladores, como UserController e EventController, são responsáveis pela lógica de negócios, centralizando operações e facilitando o fluxo de dados dentro do sistema.

O diagrama de casos de uso destaca as interações de usuários e administradores, incluindo funcionalidades como compra, cancelamento e avaliação de ingressos, e operações administrativas, como o cadastro e o gerenciamento de eventos. Essa documentação técnica revelou-se fundamental para guiar o desenvolvimento, proporcionando uma visão geral das responsabilidades de cada classe e do fluxo de interação.

2.3. Escolhas de Tecnologias: Java e JSON

A escolha de Java como linguagem de programação foi orientada por sua robustez, vasto ecossistema de bibliotecas e suporte para manipulação de dados e exceções. Java se mostrou adequado para garantir uma estrutura de código organizada e orientada a objetos, necessária

para modularizar e organizar o projeto. Java foi mantido devido à sua maior compatibilidade com ferramentas de teste e automação.

A persistência foi implementada com arquivos JSON, que, embora limitado, foi escolhido pela facilidade de implementação e compatibilidade com bibliotecas JSON em Java. JSON foi preferido por ser mais simples e flexível, ideal para um sistema em desenvolvimento inicial, onde a manipulação e a leitura de arquivos estruturados eram requisitos principais.

2.4. Estratégia de Tratamento de Erros

A estratégia de tratamento de erros do sistema consistiu na criação de exceções personalizadas que geram mensagens específicas para os usuários e logs detalhados para desenvolvedores. Para facilitar a distinção entre mensagens amigáveis e logs técnicos, foi definida uma hierarquia de exceções derivada de uma classe base, TicketSalesManagerException. Exceções como UserNotAuthorizedException, TicketAlreadyExistsException e PurchaseException fornecem mensagens compreensíveis aos usuários, enquanto para os desenvolvedores há mensagens adicionais que facilitam a identificação de erros.

2.5. Persistência de Dados

A camada de persistência foi implementada usando arquivos JSON gerenciados por uma camada DAO, que encapsula o acesso a dados e oferece métodos de leitura e escrita. Cada entidade persistente, como UserDAO e EventDAO, manipula dados utilizando métodos genéricos oferecidos pela classe DataAccessObject<T>. Esse padrão facilitou as operações de leitura e escrita, oferecendo consistência de dados durante a execução do sistema. Os métodos writeData e readData garantem que as informações, como dados de usuários, eventos e compras, sejam mantidas corretamente em arquivos JSON, o que proporciona uma boa manutenção de estado.

2.6. Testes

Os testes unitários, desenvolvidos com JUnit, garantiram uma validação completa do sistema, incluindo as operações de cadastro, atualização e exclusão de dados, bem como de funcionalidades mais complexas, como a compra e cancelamento de ingressos. Os testes abrangem desde a validação de métodos básicos até a execução de fluxos de operação complexos, como a compra de ingressos e o tratamento de exceções. Durante os testes, verificou-se que o método cancelTicket, por exemplo, lança a exceção TicketNotCancelableException ao tentar cancelar um ingresso já inativo. A cobertura dos testes unitários foi fundamental para a qualidade do sistema e ajudou a evitar problemas de regressão, proporcionando segurança para futuras atualizações do código.

3. Conclusões

O sistema de gerenciamento de eventos desenvolvido neste trabalho demonstrou ser uma solução eficaz para a gestão de eventos e a venda de ingressos. A modularidade, a utilização de testes unitários e a escolha de tecnologias adequadas contribuíram para a qualidade e a robustez do sistema. Como trabalhos futuros, sugere-se a realização de um estudo

comparativo entre diferentes tecnologias de banco de dados para identificar a mais adequada para o sistema, a implementação de um mecanismo de recomendação de eventos e a integração com plataformas de pagamento.

3.1. Resumo dos Principais Resultados

O sistema de gerenciamento de eventos e vendas de ingressos cumpriu os principais objetivos, oferecendo funcionalidades completas de cadastro e gerenciamento de eventos, controle de assentos, compra e cancelamento de ingressos, além de persistência de dados e confirmação de compras via e-mail. Essas funcionalidades foram projetadas para melhorar a eficiência e segurança nas operações de venda de ingressos, atendendo aos requisitos definidos pela empresa de entretenimento.

3.2. Contribuições

O projeto trouxe contribuições relevantes, como o uso de JSON para persistência de dados e a implementação de uma camada DAO para organização e manutenção da integridade dos dados. Além disso, o uso de exceções personalizadas e o tratamento de mensagens específicas para usuários e desenvolvedores proporcionaram uma comunicação mais clara e organizada do sistema.

3.3. Aprendizados e Desafios

O desenvolvimento deste sistema proporcionou uma rica experiência de aprendizado, abrangendo diversas áreas da engenharia de software. A modularização do sistema, aliada ao uso de exceções personalizadas, exigiu um planejamento cuidadoso da estrutura de classes e do fluxo de dados. A integração entre as camadas de persistência e controle de exceções apresentou desafios significativos, que foram superados com a implementação de métodos de verificação de consistência e um tratamento de erros mais robusto.

A experiência com testes unitários foi fundamental para garantir a qualidade do código e identificar possíveis falhas precocemente. A manipulação de JSON e a utilização do Java como linguagem de programação foram consolidadas, demonstrando sua eficiência para o desenvolvimento de sistemas de médio porte. Além disso, o projeto permitiu aprimorar habilidades em:

- **Encapsulamento e coesão:** A organização do código em módulos bem definidos e com responsabilidades claras contribuiu para a manutenibilidade e extensibilidade do sistema.
- **Gerenciamento de fluxos complexos:** A implementação de funcionalidades como a compra e o cancelamento de ingressos exigiu a gestão de diversos estados e a consideração de diferentes cenários, o que aprimorou a capacidade de lidar com complexidade.
- **Segurança e eficiência:** A preocupação com a segurança dos dados e a otimização do código foram aspectos importantes durante todo o desenvolvimento.
- **Modularização:** A divisão do sistema em módulos bem definidos facilitou o desenvolvimento, a manutenção e a identificação de problemas.

- **Tratamento de exceções:** A implementação de exceções personalizadas e um sistema de tratamento de erros robusto garantiram a robustez do sistema e a entrega de mensagens de erro claras para o usuário.
- **Testes unitários:** A cobertura de testes unitários foi fundamental para garantir a qualidade do código e identificar problemas de forma precoce.
- **Persistência de dados:** O uso de JSON para a persistência de dados demonstrou ser uma solução eficiente para este tipo de aplicação.
- **Java:** A linguagem Java se mostrou adequada para o desenvolvimento de sistemas de médio porte, oferecendo uma ampla gama de recursos e ferramentas.

Em resumo, este projeto proporcionou um aprendizado significativo em diversas áreas da engenharia de software, consolidando conhecimentos e habilidades essenciais para o desenvolvimento de sistemas robustos e escaláveis.

3.4. Trabalho Futuro

Para aprimorar o sistema, a migração para um banco de dados relacional ou NoSQL é uma proposta viável, melhorando a capacidade de consulta e performance. A implementação de design patterns mais específicos, como o padrão Observer para gerenciar estados de eventos, também poderia melhorar a coesão e reduzir o acoplamento entre os módulos. Além disso, um sistema de logs mais robusto e a implementação de testes de integração aumentariam a confiabilidade e segurança da aplicação.

4. Bibliografia

Para a realização deste projeto, foram fundamentais fontes de referência de temas relacionados à programação, arquitetura de sistemas e controle de versões, entre outras. Destacam-se as seguintes:

4.1. Livros

- BLOCH, J. **Effective Java**. Addison-Wesley Professional, 2008.
- GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design Patterns: Elements of Reusable Object-Oriented Software**. Addison-Wesley Professional, 1994.
-
- MARTIN, R. C. **Clean Code: A Handbook of Agile Software Craftsmanship**. Prentice Hall, 2008.

4.2. Documentação

- ORACLE. **Documentação Oficial do Java**. Disponível em: https://docs.oracle.com/en/java/. Acesso em: [data da consulta].
- Eclipse Foundation. **Jakarta Mail API**. Disponível em: https://eclipse-ee4j.github.io/mail/. Acesso em: 15/10/2024.

4.3. Outros

- STACK OVERFLOW. **Stack Overflow**. Disponível em: https://stackoverflow.com/. Acesso em:

15/10/2024.