linha horizontal

**Universidade Federal de Goiás**

***Disciplina***: Sistemas Distribuídos

***Departamento***: Instituto de Informática

***Integrantes***: Fabio Ofugi Mikami, Paulo Victor Passos, Rafael Oliveira de Melo, Rilbert Fernando Teixeira Santos, Vinícius Correia Soares

TicketSafe

**Serviço para gerenciamento de reservas de ingressos**

# 1*. VISÃO GERAL*

# O serviço de gerenciamento de ingressos é um sistema back-end distribuído que fornece funcionalidades para a venda, reserva e gerenciamento de ingressos para eventos. Este sistema é projetado para operar em um ambiente não web, permitindo integração com aplicativos móveis ou sistemas de terceiros, utilizando princípios de sistemas distribuídos como escalabilidade, disponibilidade e tolerância a falhas.

# 1.1 CONTEXTO

# Com o aumento da demanda por eventos e a necessidade de soluções que possam ser integradas a diferentes plataformas, a construção de um sistema back-end robusto e distribuído é crucial. Este projeto tem como objetivo desenvolver uma arquitetura que otimize o desempenho e a confiabilidade do gerenciamento de ingressos.

# 

# 1.2 PÚBLICO-ALVO

# 

# *Organizadores de Eventos*: Empresas ou indivíduos que gerenciam eventos e precisam de uma plataforma para controlar ingressos.

# *Consumidores*: Usuários que compram ingressos por meio de aplicativos ou sistemas de terceiros.

# *Agências de Vendas*: Empresas que atuam como intermediárias na venda de ingressos.

# 2. *OBJETIVOS*

# 2.1 OBJETIVO GERAL

# Desenvolver um sistema back-end distribuído de gerenciamento de ingressos que permita a venda e reserva de ingressos de forma eficiente, escalável e segura, sem a necessidade de uma interface web.

# 2.2 OBJETIVOS ESPECÍFICOS

# 1. *Gerenciamento de Eventos*: Permitir que organizadores criem, atualizem e excluam eventos.

# 2. *Processamento de Ingressos*: Implementar funcionalidades para venda e reserva de ingressos com alta disponibilidade.

# 

# 

# 3. *Autenticação e Autorização*: Implementar um sistema seguro de autenticação.

# 4. *Relatórios e Análises*: Fornecer relatórios sobre vendas e frequência de eventos, aproveitando dados distribuídos.

# 5. *Integração de Pagamentos*: Integrar métodos de pagamento seguros.

# 6. *Escalabilidade Horizontal*: Garantir que o sistema possa ser ampliado facilmente para lidar com a demanda crescente.

# 7. *Tolerância a Falhas*: Implementar estratégias que assegurem o funcionamento do sistema mesmo diante de falhas.

# 3.*ESPECIFICAÇÕES*

# 3.1 ARQUITETURA DO SISTEMA

# 3.1.1 MODELO DE MICROSSERVIÇOS

# O sistema será implementado usando uma arquitetura de microserviços, permitindo que diferentes partes do sistema sejam desenvolvidas, implantadas e escaladas de forma independente.

# Principais Microserviços:

# *Eventos*: Gerenciamento de eventos.

# *Ingressos*: Venda e reserva de ingressos.

# *Usuários*: Autenticação e gestão de usuários.

# *Pagamentos*: Processamento de transações financeiras.

# *Relatórios*: Geração de relatórios analíticos.

# 3.1.2 COMUNICAÇÃO ENTRE MICROSSERVIÇOS

# *REST APIs*: Os microserviços se comunicarão através de APIs RESTful, proporcionando um interface padronizada para interações com aplicativos ou sistemas clientes.

# *Mensageria Assíncrona*: Utilização de sistemas de mensageria como RabbitMQ ou Apache Kafka para comunicação assíncrona, permitindo desacoplamento entre serviços e garantindo que as mensagens sejam entregues mesmo em caso de falhas temporárias.

# 3.2 GESTÃO DE DADOS DISTRIBUÍDOS

# 3.2.1 BANCO DE DADOS

# *Banco de Dados Relacional*: PostgreSQL para dados estruturados.

# *Banco de Dados NoSQL*: MongoDB para flexibilidade e armazenamento de dados não estruturados.

# 3.2.2 REPLICAÇÃO E CONSISTÊNCIA

# *Replicação de Dados*: Implementar replicação de banco de dados para garantir alta disponibilidade e recuperação de desastres.

# *Consistência Eventual*: Aplicar princípios de consistência eventual em operações que podem tolerar latência, como a atualização de dados em serviços de relatórios.

# 3.3 ESCALABILIDADE E DESEMPENHO

# 3.3.1 ESCALABILIDADE HORIZONTAL

# *Containers*: Utilizar Docker para empacotar microserviços e Kubernetes para orquestração, facilitando a escalabilidade horizontal.

# *Balanceamento de Carga*: Implementar balanceadores de carga para distribuir as solicitações entre instâncias de microserviços, garantindo que nenhuma instância fique sobrecarregada.

# 3.3.2 ESTRATÉGIAS EM CACHE

# *Cache em Memória*: Utilizar Redis ou Memcached para armazenar dados frequentemente acessados, melhorando o desempenho e reduzindo a carga no banco de dados.

# 3.4 TOLERÂNCIA A FALHAS

# 3.4.1 MECANISMOS DE RECUPERAÇÃO

# *Circuit Breaker*: Implementar padrões como Circuit Breaker para evitar falhas em cascata entre microserviços.

# *Re-tentativas*: Utilizar lógica de re-tentativa em chamadas de serviços externos.

# 3.4.2 MONITORAMENTO E RESILIÊNCIA

# *Monitoramento Contínuo*: Utilizar Prometheus e Grafana para monitorar a saúde dos serviços, registrando métricas de desempenho e detectando anomalias.

# *Logging*: Implementar um sistema de logging centralizado usando ELK Stack (Elasticsearch, Logstash, Kibana) para facilitar a análise de logs.

# 4. *SEGURANÇA*

# 4.1 AUTENTICAÇÃO E AUTORIZAÇÃO

# Implementar autenticação baseada em JWT, garantindo que cada microserviço valide tokens para acesso seguro.

# 4.2 PROTEÇÃO DE DADOS

# *Comunicação Segura*: Utilizar HTTPS para todas as comunicações entre serviços e com clientes.

# *Criptografia de Dados Sensíveis*: Proteger informações sensíveis, como senhas e dados de pagamento, com criptografia forte.

# 5. *TESTES*

# 5.1 ESTRATÉGIAS DE TESTES

# *Testes Unitários*: Cobrir cada microserviço com testes unitários para garantir a funcionalidade correta.

# *Testes de Integração*: Validar a interação entre microserviços.

# *Testes de Performance*: Avaliar o desempenho do sistema sob carga.

# 5.2 FERRAMENTAS DE TESTE

# *Testes Unitários*: Pytest (Python) ou Jest (Node.js).

# *Testes de Integração*: Postman ou Insomnia.

# *Testes de Performance*: JMeter ou Locust.

# 6. *MANUTENÇÃO*

# 6.1 MONITORAMENTO CONTÍNUO

# Implementar um ciclo de feedback contínuo para identificar áreas de melhoria com base em dados de uso e desempenho.

# 6.2 ATUALIZAÇÕES E MELHORIAS

# Atualizações regulares do sistema e implementação de novas funcionalidades conforme necessário.

# 

# 7. *CONCLUSÃO*

# Este projeto de sistema distribuído visa criar uma plataforma robusta para gerenciamento de ingressos que não apenas atenda às necessidades dos organizadores de eventos e consumidores, mas também seja escalável, resiliente e fácil de manter. Ao aplicar princípios de sistemas distribuídos, garantimos que o sistema será capaz de evoluir e se adaptar às mudanças no mercado.

# 

.