Título: Documento de Arquitetura – Sistema de Rifas Digitais
Autor: Marcos Morais
Data: 05/11/2025

1. Problema

- Não existe um sistema centralizado para gerenciar rifas de forma digital.
- Organizadores precisam controlar manualmente itens rifados, bilhetes e datas de sorteio.
- Falta transparência e confiabilidade para os participantes, já que não há integração com resultados oficiais da **Loteria Federal do Brasil**.

---

2. Objetivo

Construir um sistema de rifas digital que permita:

- Permita cadastrar itens rifados (carro, moto, terreno, etc).
- Gere automaticamente bilhetes numerados de acordo com a quantidade definida.
- Associe cada bilhete a um comprador/participante.
- Registre a data do sorteio vinculada à Loteria Federal.
- Ofereça transparência e auditoria dos resultados.

---

3. Hipótese

- Se o sistema oferecer flexibilidade (API ou inserção manual), então:
    - Organizadores terão menos trabalho manual e maior confiabilidade.
    - Participantes terão mais segurança e transparência.
    - O sistema poderá escalar para múltiplos itens rifados e diferentes datas de sorteio.

---

4. Solução — Visão de Alto Nível

Módulos principais:

- **Módulo de Cadastro de Rifas:** registrar item, quantidade de bilhetes, data do sorteio, regras.
- **Módulo de Bilhetes:** geração automática de bilhetes numerados e controle de estados (disponível / reservado / vendido).
- **Módulo de Participantes:** cadastro de compradores, verificação básica (CPF) e histórico de bilhetes.
- **Módulo de Pagamentos/Transações:** integração com gateways (Pix, cartão), registro e conciliação.
- **Módulo de Sorteio / Apuração:** integração com API da Loteria Federal (quando habilitado) ou entrada manual do número sorteado.
- **Módulo de Relatórios / Auditoria:** histórico completo de rifas, transações, bilhetes e apurações com assinatura temporal.
  Frontend (Web / Mobile): interfaces separadas para Organizadores (Admin) e Participantes (compra e acompanhamento).
- **Infra / Observabilidade:** métricas, logs estruturados, rastreabilidade e backups

Fluxo simplificado:

1. Organizador cadastra o item rifado e define quantidade de bilhetes + data do sorteio.
2. Sistema gera bilhetes numerados e disponibiliza para venda.
3. Participantes compram bilhetes e recebem confirmação.
4. Na data do sorteio, sistema consulta resultado da Loteria Federal.
5. Número sorteado é cruzado com bilhetes → vencedor é identificado.
6. Relatório é publicado para transparência.

Documento de Arquitetura – Sistema de Rifas

1. Visão Geral

Sistema para gerenciamento de rifas digitais, integrando itens rifados, geração de bilhetes numerados, cadastro de participantes e apuração automática baseada nos resultados da Loteria Federal do Brasil.

2. Requisitos

2.1 Funcionais

- Cadastrar itens rifados (carro, moto, terreno, etc).
- Definir quantidade de bilhetes e data do sorteio.
- Gerar bilhetes numerados automaticamente.
- Associar bilhetes a participantes (compra/reserva).
- Consultar resultados da Loteria Federal e identificar vencedor.
- Emitir relatórios de transparência (histórico de rifas e vencedores).

2.2 Não Funcionais

- Segurança: autenticação e autorização para organizadores e participantes.
- Escalabilidade: suportar múltiplas rifas simultâneas.
- Disponibilidade: sistema acessível 24/7.
- Auditoria: rastreabilidade de transações e sorteios.
- Usabilidade: interface simples e responsiva (web e mobile).

5. Casos de Uso (UML simplificado)

Atores: Administrador (Organizador), Participante, Sistema, Fonte Externa (API Loteria Federal)

Casos de Uso Principais:

- **UC01 – Cadastrar Rifa** — Administrador: cria rifa com item, quantidade de bilhetes, preço e data.
- **UC02 – Gerar Bilhetes** — Sistema: gera bilhetes numerados automaticamente.
- **UC03 – Comprar Bilhete** — Participante: seleciona bilhete, paga, recebe confirmação.
- **UC04 – Configurar Origem do Sorteio** — Administrador: define API da Loteria ou manual.
- **UC05 – Executar Sorteio** — Sistema: consulta API ou recebe manualmente o número sorteado.
- **UC06 – Identificar Bilhete Vencedor** — Sistema: cruza número sorteado com bilhetes emitidos.
- **UC07 – Emitir Relatório de Transparência** — Sistema: publica relatório com origem do sorteio e prova de apuração.

---

6. Modelo de Dados (resumo)

Entidades principais e atributos (resumo):

- **Rifa:** id, título, descrição, item, preco_bilhete, quantidade_bilhetes, data_sorteio, concurso, usar_api_loteria (bool), status.
- **Bilhete:** id, numero (único por rifa), estado (disponível/reservado/vendido), rifa_id, participante_id (opcional).
- **Participante:** id, nome, cpf, email, telefone, endereco (opcional).

- **Transacao:** id, rifa_id, participante_id, valor_total, status (pendente/pago/cancelado), relacionamento com bilhetes (N..N).
- **Apuracao:** id, rifa_id, fonte (API/Manual), concurso, numero_sorteado, bilhete_vencedor_id, participante_vencedor_id, data_registro.

Relacionamentos:

- **Rifa 1..N Bilhetes**
- **Participante 1..N Bilhetes (via compra)**
- **Participante 1..N Transações**
- **Rifa 1..1 Apuração** (uma apuração vinculada à rifa)

---

7. Quadro de Tecnologias Possíveis (decisão adotada: Django + PostgreSQL + Next.js)

Tecnologia	Uso no Sistema	Pontos Positivos	Pontos Negativos
Django (Python)	Backend/API + Painel Administrativo	- Framework completo (ORM, autenticação, admin pronto) - Produtividade alta - Boa integração com Django REST Framework	- Performance inferior em cenários de alta concorrência - Menos flexível para arquiteturas altamente customizadas
Node.js (JavaScript)	Backend/API REST	- Alta performance em operações I/O - Ecossistema rico (npm) - Comunidade ativa	- Menos adequado para operações CPU-intensivas - Requer boas práticas para evitar problemas de concorrência
.NET Core (C#)	Backend/API REST	- Performance robusta - Suporte corporativo da Microsoft - Ferramentas maduras de segurança e logging	- Curva de aprendizado maior para quem não vem do mundo Microsoft - Menos popular em startups comparado a Node.js
Java Spring Boot	Backend/API REST	- Framework consolidado e estável - Excelente para sistemas complexos e corporativos - Grande comunidade	- Verbosidade do Java - Consumo de recursos maior que alternativas mais leves
Next.js (React)	Frontend Web	- Renderização híbrida (SSR/SSG) - Excelente para SEO - Integração nativa com React - Ótimo para aplicações escaláveis	- Complexidade maior que React puro - Requer conhecimento de SSR/SSG para aproveitar bem

React	Frontend Web	- Biblioteca flexível e popular - Grande ecossistema de componentes - Comunidade ativa	- Necessidade de configurar várias ferramentas adicionais (roteamento, estado) - Curva de aprendizado para iniciantes
Flutter	Frontend Mobile	- Código único para iOS e Android - Alta performance (compilado nativo) - UI moderna e rica	- Aplicativos podem ficar pesados - Comunidade menor que React Native
PostgreSQL	Banco de Dados Relacional	- Suporte avançado a queries complexas - Estabilidade e confiabilidade - Open source	- Configuração inicial pode ser mais complexa - Menos flexível para dados não estruturados
MongoDB	Banco de Dados NoSQL	- Flexível para dados não estruturados - Escalabilidade horizontal fácil - Modelo de documentos intuitivo	- Menos adequado para transações complexas - Consistência eventual em clusters distribuídos

---

8. Modelo Lógico (Arquitetura de Componentes)

Camada	Responsabilidade	Tecnologias sugeridas
Apresentação	Interface web/mobile para organizadores e participantes	React, Angular, Flutter
Serviços (API)	Exposição de endpoints REST para operações de rifas	Node.js, .NET Core, Java Spring
Negócio	Regras de geração de bilhetes, validação de sorteios	Classes de domínio, serviços
Integração	Consulta de resultados da Loteria Federal	API Caixa Econômica Federal
Persistência	Armazenamento de itens, bilhetes, participantes	PostgreSQL ou MongoDB
Infraestrutura	Deploy em nuvem, escalabilidade, logs	AWS, Azure ou GCP

8. Modelo Físico (Infraestrutura)

- **Frontend:** hospedado em CDN (CloudFront, Azure Front Door).
- **Backend:** cluster de containers (Docker + Kubernetes).
- **Banco de Dados:** instância gerenciada (RDS, CosmosDB).
- **Integração:** serviço de integração com API da Loteria Federal.
- **Monitoramento:** Prometheus + Grafana, logs centralizados (ELK Stack).

9. Considerações Arquiteturais

- **Flexibilidade:** operar com ou sem API da Loteria; origem do número visível no relatório.
- **Transparência:** relatórios com carimbo temporal e origem da apuração.
- **Escalabilidade:** particionar rifas de alto volume, usar caches e filas para controle de concorrência.
- **Segurança / LGPD:** proteção de dados pessoais (CPF, contato), criptografia em repouso/transporte, consentimento explícito.
- **Concorrência em vendas:** usar lock otimista/pessimista (Redis + transações DB) para evitar dupla venda de bilhetes.
- **Auditoria:** registrar eventos imutáveis (apuração, mudança de status) com timestamps.
