📘 ESPECIFICAÇÃO DO SISTEMA

Sistema Híbrido de Bingo (Digital + Presencial)

---

1. Visão Geral

1.1 Objetivo

Desenvolver um **sistema híbrido de bingo para eventos**, permitindo:

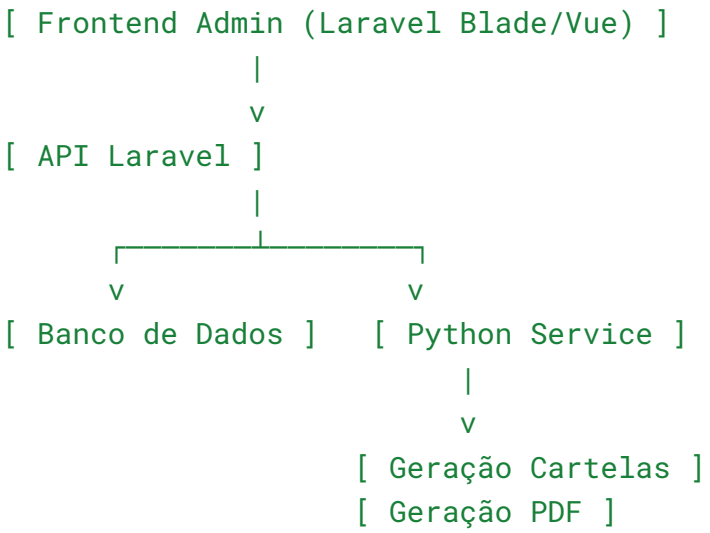
- Participação **presencial (cartelas impressas)**
- Participação **digital (web/app)**
- Geração automática de cartelas
- Controle de sorteios
- Conferência de prêmios
- Auditoria completa do evento

1.2 Características-chave

- Geração de **cartelas únicas por rodada**
- Cada cartela possui **5 subcartelas (5 rodadas)**
- Integração Laravel (admin) + Python (geração/PDF)
- Sistema determinístico e auditável
- Escalável (2.000+ cartelas por evento)

---

2. Arquitetura Geral



---

3. Tecnologias

3.1 Backend Administrativo

- Laravel 10+
- PHP 8.2+
- Blade ou Vue
- MySQL / PostgreSQL
- Redis (opcional)

3.2 Serviço de Geração

- Python 3.11+
- FastAPI ou CLI
- ReportLab (PDF)
- hashlib / secrets

3.3 Comunicação

- REST API (Laravel ↔ Python)

- WebSocket (Laravel – sorteio ao vivo)

---

4. Módulos do Sistema

🔹 MÓDULO 1 – Autenticação e Perfis

4.1 Tipos de usuários

Perfil	Descrição
Admin	Criação e gestão de eventos
Operador	Conduz sorteios
Auditor	Consulta relatórios
Jogador Digital	Marca cartelas online

4.2 Funcionalidades

- Login seguro
- Controle de permissões
- Log de ações administrativas

🔹 MÓDULO 2 – Criação de Evento

5.1 Interface (Admin)

Campos:

- Nome do evento
- Data/hora

- Quantidade de cartelas
- Quantidade de rodadas (fixo: 5)
- Tipo de bingo (75 bolas)
- Tipo de participação:
    - Digital
    - Presencial
    - Híbrido
- Descrição

Botões:

- Salvar rascunho
- Gerar cartelas
- Finalizar evento

5.2 Regras

- Evento só pode ser iniciado após geração das cartelas
- Evento gera um **seed global único**
- Evento possui status:
    - draft
    - generated
    - running
    - finished

🔹 MÓDULO 3 – Geração de Cartelas (Python)

6.1 Requisitos

- Gerar N cartelas
- Cada cartela contém 5 subcartelas
- Unicidade garantida por rodada
- FREE no centro
- Determinístico via seed

6.2 Algoritmo (conceitual)

Seed:

```
HASH(event_id + rodada + indice_cartela + secret)
```

Geração:

- Colunas B/I/N/G/O
- Hash único da subcartela
- Persistência do hash

6.3 Integração

Laravel envia:

```
{
  "event_id": "uuid",
  "total_cards": 2000,
  "rounds": 5,
  "seed": "hash"
}
```

Python retorna:

```
{
  "status": "ok",
  "generated": 10000
}
```

🔷 MÓDULO 4 – Geração e Impressão de PDF

7.1 Layout da Cartela

Cada cartela:

- Identificador único
- QR Code
- 5 subcartelas (uma por rodada)
- Nome do evento
- Regras resumidas

7.2 Funcionalidades

- Geração em lote
- Download ZIP
- Impressão A4 / A5
- Margens configuráveis

🔹 MÓDULO 5 – Sorteio de Números

8.1 Interface do Operador

- Botão “Sortear número”
- Histórico em tempo real
- Exibição em telão
- Pausar / Encerrar rodada

8.2 Regras

- Sorteio único por número
- Sequência auditável
- Registro de timestamp

8.3 Comunicação

- WebSocket para jogadores digitais
- Tela pública (modo apresentação)

🔹 MÓDULO 6 – Conferência de Bingo

9.1 Modos de Conferência

Digital

- Jogador solicita bingo
- Sistema valida automaticamente

Presencial

- Operador escaneia QR Code
- Sistema valida subcartela da rodada

9.2 Validação

- Regeração da subcartela
- Aplicação do padrão da rodada
- Conferência contra números sorteados

🔹 MÓDULO 7 – Relatórios e Auditoria

10.1 Relatórios

- Cartelas geradas
- Números sorteados
- Vencedores por rodada
- Tentativas inválidas

10.2 Auditoria

- Reprodutibilidade total
- Log por usuário
- Exportação CSV/PDF

🔹 MÓDULO 8 – Consulta de Cartelas

11.1 Admin

- Listar cartelas
- Visualizar subcartelas
- Filtrar por rodada

11.2 Jogador

- Visualização digital
- Marcação automática/manual

🔹 MÓDULO 9 – Segurança

- Seed nunca exposto
- Hash criptográfico
- Rate limit
- Logs imutáveis
- Validação 100% no backend

🔹 MÓDULO 10 – Escalabilidade

- Geração em batch
- Redis para sorteios
- Workers Laravel

- Serviço Python isolado

---

12. Estados do Evento (Diagrama)

`DRAFT → GENERATED → RUNNING → FINISHED`

---

13. Logs do Sistema

- Criação de evento
- Geração de cartelas
- Sorteios
- Premiações
- Ações administrativas

---

14. Roadmap de Implementação

Fase 1

- Core do sistema
- Geração de cartelas
- PDF

Fase 2

- Sorteio ao vivo
- Validação digital

Fase 3

- Relatórios
- Auditoria
- Escala

---

15. Conclusão

Este sistema:

- É **robusto**
- É **auditável**
- Funciona **presencial e digital**
- Pode virar **produto comercial**
- Pode atender eventos grandes sem risco

Especificação do desenvolvimento do sistema

Etapa 1 – Diagrama de Banco de Dados

1. Objetivo desta etapa

Definir o **modelo de dados completo** do sistema de bingo híbrido (digital + presencial), garantindo:

- Integridade
- Auditabilidade
- Escalabilidade
- Suporte a múltiplos eventos
- Unicidade de subcartelas por rodada

Este diagrama serve como **base estrutural** para as próximas etapas (OpenAPI, layout e código).

---

2. Visão geral do modelo

Entidades principais:

- Usuários e permissões
- Eventos de bingo
- Cartelas
- Subcartelas (rodadas)
- Números das subcartelas
- Sorteios
- Validações de bingo
- Logs e auditoria

Relacionamentos-chave:

- Um evento possui muitas cartelas
- Uma cartela possui 5 subcartelas
- Cada subcartela pertence a uma rodada
- Um evento possui vários sorteios
- Um bingo validado sempre referencia uma subcartela

---

3. Tabelas do sistema

3.1 users

Usuários do sistema (admin, operador, auditor, jogador digital).

Campo	Tipo	Observação
id	uuid	PK
name	varchar	
email	varchar	unique
password	varchar	hash
role	enum	admin, operador, auditor, jogador
created_at	timestamp	

updated_at	timestamp

3.2 events

Evento de bingo.

Campo	Tipo	Observação
id	uuid	PK
name	varchar	
description	text	
event_date	datetime	
total_cards	int	ex: 2000
total_rounds	int	fixo = 5
seed	varchar	nunca exposto
status	enum	draft, generated, running, finished
created_by	uuid	FK users
created_at	timestamp	
updated_at	timestamp	

3.3 cards

Cartela principal do jogador.

Campo	Tipo	Observação
id	uuid	PK
event_id	uuid	FK events
card_index	int	sequencial no evento
qr_code	varchar	identificador público
created_at	timestamp	

Constraint:

- UNIQUE(event_id, card_index)

3.4 subcards

Subcartelas (uma por rodada).

Campo	Tipo	Observação
id	uuid	PK
card_id	uuid	FK cards
event_id	uuid	redundância intencional
round_number	tinyint	1 a 5
hash	varchar	hash da subcartela
created_at	timestamp	

Constraints:

- UNIQUE(event_id, round_number, hash)
- UNIQUE(card_id, round_number)

3.5 subcard_numbers

Números da subcartela (opcional, mas recomendado para auditoria e PDF).

Campo	Tipo	Observação
id	bigint	PK
subcard_id	uuid	FK subcards
row	tinyint	0–4
col	tinyint	0–4
value	varchar	número ou FREE

3.6 draws

Registro dos números sorteados.

Campo	Tipo	Observação
id	uuid	PK
event_id	uuid	FK events
round_number	tinyint	rodada atual
number	tinyint	1–75
draw_order	int	sequência
drawn_at	timestamp	

Constraints:

- UNIQUE(event_id, number)

3.7 bingo_claims

Tentativas de bingo (válidas ou não).

Campo	Tipo	Observação
id	uuid	PK
event_id	uuid	FK events
subcard_id	uuid	FK subcards
claimed_by	uuid	FK users (nullable p/ presencial)
is_valid	boolean	resultado
validated_at	timestamp	
created_at	timestamp	

3.8 winners

Registro oficial de vencedores.

Campo	Tipo	Observação
id	uuid	PK
event_id	uuid	FK events
round_number	tinyint	
subcard_id	uuid	FK subcards
prize_description	varchar	
awarded_at	timestamp	

Constraint:

- UNIQUE(event_id, round_number)

3.9 system_logs

Logs de auditoria.

Campo	Tipo	Observação
id	bigint	PK
user_id	uuid	FK users
action	varchar	
reference_type	varchar	event, card, draw
reference_id	uuid	
metadata	json	
created_at	timestamp	

---

4. Relacionamentos (resumo)

- users 1:N events
- events 1:N cards

- cards 1:N subcards
- subcards 1:N subcard_numbers
- events 1:N draws
- subcards 1:N bingo_claims
- events 1:N winners

---

5. Garantias do modelo

✔ Unicidade de subcartelas por rodada
✔ Auditoria completa
✔ Reprodutibilidade via seed
✔ Suporte a bingo físico e digital
✔ Escalabilidade para grandes eventos

---

6. Observações técnicas importantes

- A redundância de event_id em subcards é proposital para facilitar queries
- O hash da subcartela é a base da unicidade
- subcard_numbers pode ser omitida se regeneração on-the-fly for usada

---

7. Próxima etapa

➡ Etapa 2 – Contrato OpenAPI (Laravel ↔ Python + Frontend)

Nesta etapa definiremos:

- Endpoints
- Payloads
- Autenticação
- Fluxos de geração e validação

A **Etapa 1 – Diagrama de Banco de Dados** foi criada no canvas com uma **modelagem completa, consistente e pronta para produção**, cobrindo:

- Eventos
- Cartelas
- Subcartelas por rodada (com unicidade garantida)
- Sorteios
- Validação de bingo
- Vencedores

- Auditoria e logs
- Suporte total ao modelo híbrido (digital + presencial)

Essa modelagem já está pensada para:

- **Laravel (Eloquent)**
- **Integração com Python**
- **Escalabilidade**
- **Auditoria jurídica / organizacional**, se necessário

Próximo passo (Etapa 2)

➡ **Escrever o Contrato OpenAPI**, definindo:

- Endpoints do Laravel (admin, sorteio, validação)
- Endpoints do serviço Python (geração de cartelas, PDF)
- Payloads, respostas e erros
- Autenticação e segurança entre serviços

Quando quiser, diga:

> **“Vamos para a etapa 2”**

e eu começo exatamente de onde paramos.

Etapa 2 – Contrato OpenAPI

1. Objetivo

Definir o **contrato de comunicação (OpenAPI 3.0)** entre:

- Frontend ↔ Laravel API
- Laravel API ↔ Serviço Python

Garantindo:

- Integração clara
- Separação de responsabilidades
- Segurança
- Evolução independente dos serviços

---

2. Visão Geral da Arquitetura de APIs

2.1 APIs envolvidas

API	Responsável	Função
Admin API	Laravel	Gestão de eventos, sorteios, validações
Game API	Laravel	Sorteio ao vivo, validação bingo
Generator API	Python	Geração de cartelas e PDFs

---

3. Padrões Gerais

3.1 Autenticação

- Admin / Operador / Auditor:
  - JWT (Bearer Token)
- Comunicação Laravel ↔ Python:
  - API Key + assinatura HMAC

Headers padrão:

Authorization: Bearer <token>
X-Request-Id: uuid

---

4. Admin API (Laravel)

4.1 Criar Evento

POST `/api/events`

Request:

```
{
  "name": "Bingo Beneficente",
  "description": "Evento híbrido",
  "event_date": "2026-05-10T19:00:00",
  "total_cards": 2000
}
```

Response:

{

```
  "id": "uuid",
  "status": "draft"
}
```

4.2 Gerar Cartelas (dispara Python)

POST `/api/events/{event_id}/generate-cards`

Request:

```
{
  "rounds": 5
}
```

Response:

```
{
  "status": "processing",
  "job_id": "uuid"
}
```

4.3 Listar Cartelas

GET `/api/events/{event_id}/cards`

Response:

```
[
  {
    "card_id": "uuid",
    "card_index": 1,
    "qr_code": "ABC123"
  }
]
```

4.4 Visualizar Cartela (Admin)

GET `/api/cards/{card_id}`

Response:

```
{
```

```
  "card_index": 1,
  "subcards": [
    {
      "round": 1,
      "grid": [["B1","I20","FREE",...]]
    }
  ]
}
```

4.5 Iniciar Evento

POST `/api/events/{event_id}/start`

Response:


```
{
  "status": "running"
}
```


---

5. Sorteio (Game API – Laravel)

5.1 Sortear Número

POST `/api/events/{event_id}/draw`

Response:


```
{
  "number": 42,
  "order": 10
}
```


(WebSocket broadcast automático)

5.2 Listar Números Sorteados

GET `/api/events/{event_id}/draws`

---

6. Validação de Bingo

6.1 Solicitar Validação

POST `/api/bingo/claim`

Request:

```
{
  "subcard_id": "uuid"
}
```

Response:

```
{
  "is_valid": true,
  "round": 3
}
```

---

7. Generator API (Python)

7.1 Gerar Subcartelas

POST `/generator/generate`

Headers:

X-API-KEY: secret

Request:

```
{
  "event_id": "uuid",
  "seed": "hash",
  "total_cards": 2000,
  "rounds": 5
}
```

Response:

```
{
  "generated": 10000,
  "status": "ok"
```

}

7.2 Gerar PDF

POST `/generator/pdf`

Request:

```
{
  "event_id": "uuid",
  "layout": "default"
}
```

Response:

```
{
  "pdf_url": "https://..."
}
```

---

8. Códigos de Erro

Código	Significado
400	Request inválido
401	Não autenticado
403	Não autorizado
404	Recurso não encontrado
409	Conflito (unicidade)
500	Erro interno

---

9. Versionamento

- Prefixo `/api/v1`
- Mudanças incompatíveis → nova versão

---

10. Próxima etapa

➡ Etapa 3 – Desenho do layout da cartela (PDF e Digital)

Definir:

- Estrutura visual
- Distribuição das 5 subcartelas
- QR Code
- Identificação do evento

Etapa 3 – Layout da Cartela (PDF e Digital)

1. Objetivo

Definir o **layout visual e funcional da cartela de bingo**, garantindo:

- Clareza para o jogador presencial
- Usabilidade no digital
- Facilidade de conferência
- Compatibilidade com impressão
- Identificação inequívoca de cartela, rodada e evento

Este layout será utilizado pelo **serviço Python (PDF)** e pelo **frontend digital**.

---

2. Conceito geral da cartela

Cada cartela representa **1 jogador** e contém:

- Identificação única
- **5 subcartelas independentes** (Rodadas 1 a 5)
- QR Code para conferência digital
- Informações do evento

Total por cartela:

- 5 subcartelas (5x5)
- 125 posições (incluindo FREE)

---

3. Formatos suportados

3.1 Impressão

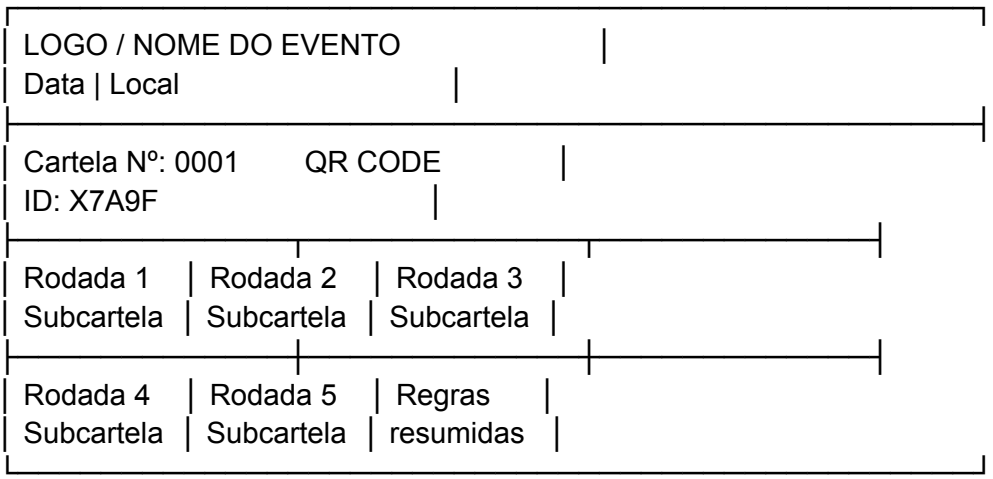
- A4 (210 × 297 mm) – padrão
- A5 (148 × 210 mm) – opcional
- Margens mínimas: 10 mm
- Impressão P&B ou colorida

3.2 Digital

- Desktop
- Tablet
- Mobile
- Layout responsivo

---

4. Estrutura do layout (A4 – referência)



---

5. Cabeçalho da cartela

Elementos obrigatórios:

- Nome do evento
- Logo (opcional)
- Data e local

Regras:

- Fonte legível (mín. 10pt)
- Alto contraste

---

6. Identificação da cartela

Campos

- Número da cartela (sequencial no evento)
- ID único (hash curto ou base32)
- QR Code

QR Code

Conteúdo:

```
{
  "event_id": "uuid",
  "card_id": "uuid"
}
```

Uso:

- Conferência presencial
- Acesso rápido no digital

---

7. Layout da subcartela (5x5)

Estrutura padrão

B I N G O

FREE

Regras visuais

- Título da rodada acima da subcartela
- Grade com linhas visíveis
- FREE destacado

---

8. Identificação da rodada

Cada subcartela deve conter:

- Texto: "Rodada X"
- Ícone ou cor opcional
- Separação visual clara entre subcartelas

Regra importante:

> Subcartelas **nunca se misturam visualmente**.

---

9. Área de regras resumidas

Conteúdo sugerido:

- Não rabiscar fora da cartela
- Conferência obrigatória
- QR Code para validação

Essa área pode ser omitida no digital.

---

10. Versão digital – comportamento

Funcionalidades

- Marcação automática ao sortear número
- Marcação manual opcional
- Destaque visual ao completar padrão
- Indicação da rodada ativa

Restrições

- Usuário não valida bingo localmente
- Apenas solicita validação

---

11. Acessibilidade

- Contraste AA
- Fonte mínima 12px no digital
- Suporte a leitores de tela (labels)

---

12. Geração no Python (PDF)

Requisitos técnicos:

- ReportLab
- Geração em lote
- Numeração sequencial
- QR Code embutido

Cada PDF pode conter:

- 1 cartela por página
- Ou múltiplas cartelas por página (configurável)

---

13. Observações importantes

- Layout deve ser versionado
- Mudança de layout não afeta dados
- Subcartelas sempre respeitam a rodada

---

14. Próxima etapa

➡ Etapa 4 – Início do código base (Laravel + Python)

Definir:

- Estrutura dos repositórios
- Setup inicial
- Comunicação entre serviços
- Primeiros endpoints funcionais

Etapa 4 – Código Base (Laravel + Python)

1. Objetivo da etapa

Iniciar a **base de código funcional** do sistema de bingo híbrido, garantindo:

- Separação clara de responsabilidades
- Comunicação estável entre Laravel e Python
- Estrutura preparada para escalar
- Primeiros fluxos ponta a ponta operacionais

Ao final desta etapa teremos:

- Projeto Laravel inicializado e organizado
- Serviço Python ativo
- Integração funcional entre os dois

---

2. Estrutura de repositórios

2.1 Organização recomendada

```
bingo-system/
├── backend-laravel/
│   ├── app/
│   ├── database/
│   ├── routes/
│   ├── config/
│   └── docker/
│
├── generator-python/
│   ├── app/
│   │   ├── main.py
│   │   ├── bingo_generator.py
│   │   ├── pdf_generator.py
│   │   └── security.py
│   ├── requirements.txt
│   └── Dockerfile
│
└── docs/
    ├── etapa-1-diagrama-banco.md
    ├── etapa-2-openapi.md
    └── etapa-3-layout-cartela.md
```

---

3. Backend Laravel – Base

3.1 Setup inicial

- Laravel 10+
- PHP 8.2+
- Sanctum ou JWT
- Queue habilitada
- Redis (opcional)

3.2 Migrations iniciais

Criar migrations para:

- users
- events
- cards
- subcards
- subcard_numbers
- draws
- bingo_claims
- winners
- system_logs

Todas baseadas **exatamente no diagrama da Etapa 1**.

3.3 Models principais

- User
- Event
- Card
- Subcard
- Draw
- BingoClaim
- Winner

Com relacionamentos:

- Event → cards, draws
- Card → subcards
- Subcard → numbers

3.4 Services (Laravel)

Criar camada de serviço:

- EventService
- CardGenerationService
- DrawService
- BingoValidationService

Nenhuma regra crítica deve ficar no controller.

3.5 Controllers iniciais

- EventController
- CardController
- DrawController
- BingoController

Todos seguindo o contrato OpenAPI.

---

4. Serviço Python – Base

4.1 Stack

- Python 3.11+
- FastAPI
- Uvicorn
- ReportLab

4.2 Estrutura do serviço

main.py

- Inicializa FastAPI
- Middleware de autenticação

bingo_generator.py

Responsável por:

- Gerar subcartelas determinísticas
- Garantir unicidade por rodada
- Retornar estrutura serializável

pdf_generator.py

Responsável por:

- Gerar PDF conforme layout da Etapa 3
- Inserir QR Code
- Exportação em lote

4.3 Segurança (Python)

- API Key fixa
- Validação HMAC
- Rate limit interno

---

5. Integração Laravel ↔ Python

5.1 Fluxo de geração

1. Admin clica "Gerar Cartelas"
2. Laravel dispara Job
3. Job chama API Python
4. Python gera subcartelas
5. Laravel persiste dados
6. Status do evento atualizado

5.2 Comunicação

- HTTP REST
- JSON
- Timeout controlado
- Retry automático

---

6. Primeiro fluxo funcional (MVP técnico)

6.1 Fluxo implementado

- Criar evento
- Gerar cartelas (2.000)
- Gerar PDFs

- Listar cartelas

Sem sorteio ainda (entra na fase seguinte).

---

7. Docker (opcional, recomendado)

- docker-compose
- Serviços:
    - laravel
    - mysql/postgres
    - redis
    - python-generator

---

8. Padrões de código

- PSR-12 (PHP)
- Black / PEP8 (Python)
- DTOs para requests
- Logs estruturados

---

9. Critérios de aceite da etapa 4

✔ Laravel sobe sem erro
✔ Python API responde
✔ Evento é criado
✔ Cartelas são geradas
✔ PDF é produzido
✔ Dados persistidos corretamente

---

10. Próximas etapas após a 4

- Etapa 5 – Sorteio ao vivo (WebSocket)
- Etapa 6 – Validação de bingo
- Etapa 7 – Relatórios e auditoria
- Etapa 8 – Endurecimento de segurança

---

11. Conclusão

Com esta etapa concluída, o sistema deixa de ser projeto e passa a ser **software funcional**, pronto para evolução incremental e uso real em eventos.