

Sistema de Conexões Reais (Modo Viagem + Bora) — Manual Técnico Definitivo

Documento detalhado, com especificações de APIs, fluxos matemáticos e arquitetura de dados.

Público:

engenheiros, devs e time técnico

.

CAMADA 01 — CONTEXTO E PROPÓSITO DO SISTEMA DE CONEXÕES REAIS (MODO VIAGEM + BORA)

Entrada

O **Sistema de Conexões Reais** (que integra **Modo Viagem + Bora**) é o coração da proposta do FriendApp: transformar intenção em realidade, energia em encontro, e presença digital em conexão física.

O propósito central desta camada é definir a **visão estratégica**, os **objetivos de alto nível** e o **escopo do sistema**, antes de mergulhar nos detalhes técnicos.

Esse sistema não é apenas uma funcionalidade — ele é o **ponto de diferenciação** que transforma o FriendApp em algo único frente a redes sociais tradicionais. Enquanto outras plataformas apenas **mostram conteúdo**, aqui o usuário **vive experiências reais** baseadas em seu estado vibracional, intenção e geografia.

Parte Técnica

Escopo Funcional do Sistema:

1. Modo Viagem

- Projeta a "intenção de deslocamento" do usuário para destinos futuros.
- Permite conexões energéticas prévias na cidade de destino.
- Ativa o matching vibracional com usuários, grupos e locais parceiros.

2. Bora (Encontros Espontâneos)

- Criação de grupos temáticos de encontro imediato ou planejado.
- Baseado em proximidade geográfica + intenção declarada.
- Governança via verificação documental (DUC/DCO).

3. Locais Parceiros (Integração B2B)

- Integra bares, cafés, coworkings e espaços parceiros.
- Oferece "zonas seguras" para encontros reais.
- Monetização via planos de assinatura (tiers).

Principais Dependências Técnicas:

- `autenticacao-service` : login seguro (OAuth2/JWT).
- `perfil-e-frequencia-service` : perfil energético do usuário.
- `conexoes-reais-service` : motor central de Viagem + Bora.
- `economia-service` : FriendCoins, planos Premium.
- `suporte-e-comunidade-service` : denúncias, governança.
- `partners-api` : integração com locais parceiros.

Principais Entradas do Sistema:

- Dados de intenção (`intencao_latente`)
- Estado vibracional (`vetor_frequencia`)
- Geolocalização do usuário (`lat` , `long`)
- Datas e destinos declarados
- Verificação documental (DUC/DCO)

Principais Saídas do Sistema:

- Sugestões de conexões (usuários, grupos, locais)
- Notificações de colisão vibracional
- Criação de grupos Bora em tempo real

- Sugestão de locais parceiros para encontros
-

Fechamento

Esta primeira camada define o **propósito macro**: um sistema que integra **intenção, energia e geografia** para criar encontros reais seguros e significativos.

A partir daqui, mergulharemos nas estruturas de dados e fluxos técnicos que viabilizam essa visão.

CAMADA 02 — MODELO DE ESTADOS DO USUÁRIO (MODO VIAGEM + BORA)

Entrada

Para o Sistema de Conexões Reais funcionar, é essencial entender em que **estado operacional** o usuário se encontra.

O FriendApp não trata todos os usuários da mesma forma — a IA precisa diferenciar se ele está ativo, viajando, organizando um Bora ou apenas explorando.

Esse modelo de estados é o que garante que as notificações, recomendações e experiências sejam contextuais, precisas e sem ruído.

Parte Técnica

Estados Principais:

1. Online Ativo (**active**)

- Usuário está navegando em tempo real.
- Disponível para colisões vibracionais locais.

2. Explorando (**exploring**)

- Usuário está no app mas apenas consumindo conteúdo.
- Matching passivo (recebe sugestões mas não é projetado em mapas Bora/Viagem).

3. Modo Viagem (**travel_mode**)

- Usuário declara destino e datas.
- Projeção energética para cidade de destino.
- Ativa pré-matching e colisões preditivas.

4. Bora Ativo (**bora_active**)

- Criou ou ingressou em um grupo Bora.
- Ativa geofencing para notificações hiperlocais.

5. Offline (**offline**)

- Usuário desconectado.
- Dados tratados apenas pela IA para predição (sem push).

Diagrama Simplificado de Transições:

```
stateDiagram-v2
    [*] → active
    active → exploring : Inatividade > 3min
    active → travel_mode : Ativa Modo Viagem
    active → bora_active : Entra em grupo Bora
    travel_mode → active : Viagem finalizada
    bora_active → active : Bora encerrado
    active → offline : Logout/Desconexão
    offline → active : Login
```

Estrutura Técnica:

- Tabela **user_state** :

```
CREATE TABLE user_state (
    user_id UUID PRIMARY KEY,
    state VARCHAR(20) NOT NULL,
    last_update TIMESTAMP NOT NULL,
    context JSONB
);
```

- Campos principais:

- `state` : active, exploring, travel_mode, bora_active, offline
- `context` : armazena metadados (ex: destino da viagem, id do grupo Bora).
- `last_update` : usado pela IA para calcular transições.

Dependências Diretas:

- `perfil-e-frequencia-service` → sincroniza estado vibracional.
- `conexoes-reais-service` → consome estado para determinar matching.
- `notifications-service` → envia alertas dependendo do estado.

Fechamento

O modelo de estados do usuário é a **espinha dorsal contextual** do sistema. Sem ele, a IA não consegue diferenciar se um push é oportuno, se uma sugestão faz sentido ou se a projeção energética deve ser ativada.

CAMADA 03 — FLUXO DE PROJEÇÃO ENERGÉTICA (MODO VIAGEM)

Entrada

O Modo Viagem não é apenas um "check-in antecipado".

Ele funciona como um **sistema de projeção energética digital**, onde a intenção do usuário é registrada, contextualizada e distribuída no ecossistema FriendApp para alimentar colisões preditivas e recomendações inteligentes.

Esse fluxo garante que a cidade de destino "sinta" a presença do viajante antes mesmo da chegada física.

Parte Técnica

Etapas do Fluxo:

1. Ativação do Modo Viagem

- Usuário define:
 - Cidade/Região de destino.
 - Datas estimadas (início e fim).

- Intenção vibracional da viagem (ex: lazer, espiritualidade, negócios, cura).
- Input armazenado em `travel_intentions`.

```
CREATE TABLE travel_intentions (  
  travel_id UUID PRIMARY KEY,  
  user_id UUID NOT NULL,  
  destino VARCHAR(100) NOT NULL,  
  data_inicio DATE NOT NULL,  
  data_fim DATE NOT NULL,  
  intencao JSONB,  
  created_at TIMESTAMP DEFAULT NOW()  
);
```

1. Normalização Geográfica

- Sistema converte o destino em **geohash** de precisão variável (bairro/cidade).
- Garante que projeções sejam anônimas em níveis públicos, mas precisas para matching interno.

1. Projeção Energética

- Vetor vibracional do usuário (`user_vector`) é enviado para **Aurah Kosmos**.
- A IA gera uma assinatura energética específica da viagem:

```
{  
  "user_id": "abc-123",  
  "destino_geohash": "6gkzwwgjz",  
  "periodo": ["2025-08-20", "2025-08-27"],  
  "vetor_viagem": [0.71, 0.12, -0.33, ...],  
  "intencao": ["cura", "expansao"]  
}
```

1. Armazenamento em Índice Preditivo

- Dados vão para o **Aurah Travel Index**, um banco otimizado em **ElasticSearch + Redis**.
- Suporte a consultas rápidas do tipo:
 - "Quais usuários estarão projetados em Lisboa em setembro?"
 - "Existe algum overlap vibracional entre viajantes e locais parceiros?".

1. Gatilhos de Colisão Preditiva

- Jobs assíncronos (Kafka + workers) rodam a cada hora para:
 - Detectar sobreposição de projeções (usuário A e B na mesma janela temporal).
 - Calcular afinidade energética entre viajantes e moradores da cidade.
 - Notificar potenciais colisões vibracionais.

Pseudocódigo Simplificado:

```
for viagem in travel_intentions:
    overlaps = query_travel_index(viagem.destino, viagem.data_inicio, viagem.data_fim)
    for outro in overlaps:
        score = aurah_kosmos.match(viagem.vetor_viagem, outro.vetor_viagem)
        if score > 0.82:
            notify_collision(viagem.user_id, outro.user_id)
```

Dependências Diretas:

- `perfil-e-frequencia-service` → fornece vetor energético do usuário.
- `aurah-kosmos-core` → calcula afinidade e gera projeção vibracional.
- `notifications-service` → envia alertas de colisões previstas.

Fechamento

O fluxo de projeção energética transforma o simples ato de "planejar uma viagem" em um **campo vibracional ativo**, capaz de gerar encontros antes mesmo da chegada física.

CAMADA 04 — INTEGRAÇÃO DA PROJEÇÃO ENERGÉTICA COM A AURAH KOSMOS

Entrada

Se a camada anterior estruturou **como a projeção de viagem é capturada e indexada**, aqui entramos no coração lógico:

a **IA Aurah Kosmos** como núcleo que interpreta, recalcula e evolui os vetores vibracionais projetados pelos usuários.

Sem essa integração, os dados seriam apenas registros estáticos.

Com ela, eles se tornam **entidades dinâmicas** capazes de gerar previsões, colisões vibracionais e recomendações.

Parte Técnica

1. Entrada de Dados Brutos

- Vetores de frequência (`user_vector`) vêm do serviço de perfis.
- Metadados da viagem: destino, período, intenção.
- O pacote é enviado para o módulo `aurah.travel_adapter` .

```
{
  "user_id": "u123",
  "vetor_base": [0.34, -0.11, 0.87, ...],
  "intencao": ["cura", "expansao"],
  "destino_geohash": "6gkzwgjz",
  "periodo": ["2025-08-20", "2025-08-27"]
}
```

2. Normalização pelo Núcleo Aurah

- O `aurah-core` aplica técnicas de:
 - **Redução dimensional** (PCA/autoencoders) para vetores de alta densidade.

- **Escalaonamento temporal:** peso extra é dado a viagens mais próximas no tempo.
- **Ajuste de intenção:** as intenções são convertidas em embeddings e somadas ao vetor.

3. Evolução Dinâmica

- Cada projeção **não é fixa**:
 - O vetor de viagem evolui conforme:
 - Alterações no estado vibracional do usuário no dia a dia.
 - Novas interações no FriendApp.
 - Jobs em lote recalculam a projeção a cada 24h ou sempre que houver mudança significativa de estado do usuário.

4. Afinidade Preditiva

- Aurah calcula a afinidade entre viajantes e residentes usando **similaridade de cossenos + matriz de compatibilidade de intenções**.

Fórmula simplificada:

$$\text{Score} = (\cos(\text{user_vector}, \text{outro_vector}) * 0.6) + (\text{compatibilidade_intencao}(\text{user}, \text{outro}) * 0.4)$$

$$\text{Score} = (\cos(\text{user_vector}, \text{outro_vector}) * 0.6) + (\text{compatibilidade_intencao}(\text{user}, \text{outro}) * 0.4)$$

- Thresholds definidos:
 - **>= 0.85** → Alta colisão prevista.
 - **0.70–0.84** → Potencial colisão.
 - **< 0.70** → Afinidade baixa (não dispara notificação).

5. Saídas

- Notificações em tempo real (colisões previstas).
- Listas de "pessoas com quem você pode se conectar" na cidade.
- Insights para Locais Parceiros (ex: "30 viajantes energéticos chegam semana que vem buscando experiências de cura").

Dependências Diretas

- `perfil-e-frequencia-service` → gera vetor energético base.
 - `intencao-service` → transforma intenção declarada em embedding.
 - `aurah-kosmos-core` → núcleo de matching e evolução.
 - `notifications-service` → envio de alertas preditivos.
-

Fechamento

Essa camada garante que a **projeção energética do Modo Viagem não seja apenas um registro estático**, mas um campo vivo que se adapta, evolui e **alimenta o ecossistema preditivo do FriendApp**.

CAMADA 05 — MECANISMOS DE COLISÕES PREDITIVAS E ALERTAS INTELIGENTES

Entrada

Se na **Camada 04** definimos como a **Aurah Kosmos** integra e evolui a projeção energética, aqui mostramos **como essas projeções se transformam em colisões preditivas** e, finalmente, em **alertas acionáveis para os usuários**.

O objetivo é claro: **antecipar encontros potenciais** e criar um fluxo de notificações úteis, evitando spam e mantendo relevância máxima.

Parte Técnica

1. Definição de Colisão Energética

- Uma **colisão** acontece quando dois vetores energéticos (usuário viajante + residente/destino) atingem um nível de afinidade relevante.
- A IA considera **três variáveis principais**:
 - Similaridade vetorial (cosine similarity).
 - Compatibilidade de intenção.
 - Sobreposição temporal (datas de estadia/atividade).

$$\text{Colisao} = (\cos(\text{user_vector}, \text{outro_vector}) * 0.5) + (\text{compatibilidade_intencao} * 0.3) + (\text{sobreposicao_temporal} * 0.2)$$
$$\text{Colisao} = (\cos(\text{user_vector}, \text{outro_vector}) * 0.5) + (\text{compatibilidade_intencao} * 0.3) + (\text{sobreposicao_temporal} * 0.2)$$

$$\text{Colisao} = (\cos(\text{user_vector}, \text{outro_vector}) * 0.5) + (\text{compatibilidade_intencao} * 0.3) + (\text{sobreposicao_temporal} * 0.2)$$

2. Thresholds de Ação

- **≥ 0.85** → Colisão Forte → dispara notificação imediata.
- **0.70–0.84** → Colisão Moderada → armazenada em fila, pode virar recomendação se houver contexto (ex: local parceiro comum).
- **< 0.70** → descartada.

3. Engine de Notificações Inteligentes

- O módulo `notifications-engine` recebe colisões e aplica **regras anti-spam**:
 - Máx. 3 notificações de colisão por semana por usuário.
 - Priorização por score mais alto.
 - Contexto obrigatório → exemplo:
 - "✨ Sua energia colidiu com 2 viajantes que estarão em Lisboa na mesma semana que você. Quer ver quem são?"
 - "⚡ Um grupo Bora alinhado à sua intenção de cura está surgindo perto de você."

4. Fluxo de Geração

1. Colisão identificada.
2. Armazenada no `collision-queue` (Redis).
3. Avaliada pela engine de relevância (score + contexto).
4. Convertida em:
 - **Notificação push** (alta afinidade).
 - **Sugestão no feed** (afinidade média).
 - **Insight privado** (quando o match não é mútuo, mas pode evoluir).

5. Tipos de Alertas

- **Preditivos (Pré-viagem):** "Você já está projetando energia para Madrid, 5 pessoas da sua vibração também estarão lá."
- **Em tempo real (Durante viagem/Bora):** "3 conexões autênticas entraram no mesmo grupo Bora que você."
- **Contextuais (Locais Parceiros):** "O Espaço Anahata terá um evento de cura que coincide com sua projeção vibracional."

6. Integrações

- `aurah-kosmos-core` → cálculo de colisões.
- `notifications-service` → envio multicanal (push, in-app, email opcional).
- `partners-api` → cruzamento com locais/eventos parceiros.
- `feed-service` → exibição como sugestão.

Fechamento

Essa camada transforma **dados em experiência real**.

O usuário não precisa "caçar conexões": o sistema se antecipa, mostrando apenas as **colisões relevantes** que realmente podem gerar encontros.

CAMADA 06 — MODERAÇÃO PREVENTIVA E SEGURANÇA EM TEMPO REAL (BORA)

Entrada

Nos encontros "Bora", o maior desafio não é apenas conectar pessoas com intenção vibracional, mas garantir que esses encontros sejam **seguros, justos e governados de forma saudável**.

Para investidores, isso mostra que o FriendApp pensa além da conexão: está construindo um ecossistema **sustentável, ético e responsável**, minimizando riscos legais e sociais.

Para os desenvolvedores, esta camada define **fluxos, permissões, APIs e algoritmos de moderação**, assegurando que os encontros ocorram em ambientes **seguros e bem regulados**.

Parte Técnica

◆ 1. Estrutura de Governança do Grupo Bora

- **Criador do Grupo (Owner)**

- Permissões: criar, editar descrição, definir local e horário, convidar.
- Moderação: expulsar, silenciar temporariamente, banir permanentemente.
- Limites: não pode remover denúncias feitas contra si.

- **Moderadores (Mods)**

- Nomeados pelo criador (máx. 3 por grupo).
- Permissões: silenciar e banir membros.
- Não podem excluir o criador nem alterar local/agenda.

- **Membros**

- Podem participar do chat, confirmar presença, denunciar abusos.
-

◆ 2. Ferramentas de Moderação

- **Silenciar Temporário**

- Endpoint: `POST /api/bora/group/{id}/mute/{userId}`
- Payload: `{ "duration": "10m|1h|24h" }`
- Motivo armazenado em logs de auditoria.

- **Banimento**

- Endpoint: `POST /api/bora/group/{id}/ban/{userId}`
- Payload: `{ "reason": "texto" }`
- Usuário não pode retornar ao grupo.

- **Sistema de Denúncias**

- Endpoint: `POST /api/bora/reports`
- Campos: `reporterId, targetId, groupId, reason, evidence (opcional)`
- Fluxo:
 1. Denúncia registrada no banco (Firestore + log criptografado).
 2. Triagem automática da IA de Moderação (Aurah Guardian).

3. Escalonamento para suporte humano se classificado como grave.

◆ 3. Segurança em Tempo Real

- **Botão de Alerta Vibracional**

- Presente na tela do encontro após o check-in.
- Endpoint: `POST /api/bora/alert`
- Ações:
 - Notifica moderadores do grupo.
 - Alerta equipe de segurança central.
 - Oferece opção de compartilhar geolocalização com contato de emergência.

- **Zonas de Encontro Seguras (Locais Parceiros)**

- Incentivo: +50XP no jogo da transmutação ao marcar encontro em locais parceiros verificados.
- Algoritmo da IA sugere locais parceiros como **default** nos primeiros encontros.

- **Logs de Segurança**

- Todos os eventos críticos (banimento, denúncias, alertas) registrados em logs criptografados (AES-256 + hash SHA-512).
 - Retenção: 24 meses.
-

◆ 4. APIs Envolvidas

- `POST /api/bora/group/create`
 - `POST /api/bora/group/{id}/mute`
 - `POST /api/bora/group/{id}/ban`
 - `POST /api/bora/reports`
 - `POST /api/bora/alert`
-

Fechamento

Com essa camada, os encontros Bora deixam de ser apenas **grupos espontâneos** e passam a ter um ecossistema de **segurança, governança e**

proteção em tempo real.

Isso garante confiança para os usuários e resiliência legal para os investidores.

CAMADA 07 — INTEGRAÇÃO DE REPUTAÇÃO VIBRACIONAL E SCORE DE HARMONIA (MODO VIAGEM + BORA)

Entrada

Se a Camada 06 trouxe **segurança preventiva e governança**, esta camada define **como medir e traduzir a qualidade vibracional de usuários e grupos em dados concretos**.

Isso garante que encontros **não sejam apenas possíveis, mas também saudáveis, equilibrados e sustentáveis**, evitando frustrações ou ambientes tóxicos.

Para investidores: é a lógica que transforma **energia vibracional** em **KPIs mensuráveis de engajamento e retenção**.

Para desenvolvedores: é a camada que define **fórmulas matemáticas, endpoints e integrações de reputação**.

Parte Técnica

◆ 1. Reputação Vibracional do Usuário

Cada usuário carrega um **Score de Reputação Vibracional (SRV)**, que é atualizado em tempo real.

Fatores:

- **Participação Positiva:** presença confirmada e check-in sem incidentes (+5 SRV).
- **Feedback Pós-Encontro:** avaliações recebidas (1 a 5 estrelas energéticas).
- **Denúncias Procedentes:** -15 a -50 SRV.
- **Harmonia em Grupo:** contribuição para o aumento ou redução da harmonia coletiva.

Cálculo Simplificado (0-100):

$$SRV = (MédiaFeedback * 20) + (EventosPositivos * 2) - (DenúnciasGraves * 30)$$

◆ 2. Score de Harmonia do Grupo Bora

Mede a **compatibilidade vibracional entre os membros**.

Fatores:

- **Similaridade de Intenções** (cosine similarity entre vetores de intenção).
- **Dispersão Energética** (quanto mais homogêneo, maior harmonia).
- **Histórico de Interações Positivas** entre os membros.

Cálculo (0–100):

$$ScoreHarmonia = (SimilaridadeMédia * 0.6) + (1 - VariânciaEnergia) * 0.3 + (HistóricoPositivo * 0.1)$$

Exemplo:

Se o grupo tiver membros com intenções alinhadas (85%), baixa variância vibracional (10%) e bons históricos (80%), o Score será $\approx 82/100$.

◆ 3. Visibilidade e UX

- Usuário vê **badge simplificada**:
 - 🌟 Alta Harmonia (80–100)
 - ⚖️ Harmonia em Construção (50–79)
 - ⚠️ Harmonia Instável (<50)
- Criadores podem usar esse **selo para atrair mais membros**.
- Grupos com harmonia baixa recebem **alertas internos** e recomendações da IA para melhorar o equilíbrio.

◆ 4. APIs Envolvidas

- `GET /api/reputation/{userId}` → retorna score SRV.

- `GET /api/bora/{groupid}/harmony` → retorna Score de Harmonia.
 - `POST /api/bora/feedback` → registra avaliação pós-encontro.
-

Fechamento

A integração entre **Reputação Vibracional** e **Score de Harmonia** cria um ecossistema **autorreferente e autocurativo**, onde grupos e usuários se fortalecem ou se ajustam conforme seus comportamentos.

Isso eleva a **confiança do sistema**, reduz riscos de encontros frustrantes e gera **dados sólidos** para investidores e stakeholders.

CAMADA 08 — MODELO DE MONETIZAÇÃO ESTRATÉGICA (B2C + B2B) DO SISTEMA DE CONEXÕES REAIS

Entrada

Se até aqui falamos de **experiência**, **governança** e **harmonia vibracional**, agora entramos no ponto que garante a **sustentabilidade do ecossistema**:

como o FriendApp captura valor econômico real a partir das conexões, sem quebrar a magia da experiência.

Essa camada é crucial para investidores: ela mostra que não se trata apenas de uma rede social, mas de uma **plataforma híbrida social + marketplace experiencial**.

Para desenvolvedores: define quais integrações financeiras, fluxos e tokens são necessários para suportar a operação.

Parte Técnica

◆ 1. Fluxos de Monetização B2C (Usuário Final)

- **Assinatura Premium (FriendApp+)**
 - Benefícios:
 - Ativação ilimitada do **Modo Viagem**.
 - Participação ilimitada em **grupos Bora simultâneos**.

- Selos exclusivos de reputação.
 - Alertas vibracionais antecipados.
 - Modelo:
 - R\$ 29/mês (Essencial).
 - R\$ 59/mês (Avançado).
 - R\$ 99/mês (Visionário, inclui bônus para Locais Parceiros).
 - **FriendCoins** (Economia Interna)
 - Usadas para:
 - Criar Boras especiais.
 - Desbloquear insights vibracionais premium.
 - Participar de eventos pagos dentro do ecossistema.
 - Conversão: compra em pacotes (100 FC ≈ R\$ 10).
 - **Pagamentos Isolados (Pay-Per-Use)**
 - Ex.: Ativar “Modo Viagem” em uma jornada específica (sem precisar ser Premium).
 - Ex.: Criar um Bora patrocinado e aberto para não-amigos.
-

◆ 2. Fluxos de Monetização B2B (Locais Parceiros)

- **Assinatura Mensal (SaaS)**
 - Tier Essencial: presença no mapa vibracional.
 - Tier Premium: destaque + possibilidade de criar eventos.
 - Tier Visionário: relatórios energéticos + patrocínio de experiências.
 - **Comissão em Transações**
 - Se um Local Parceiro vender ingressos, bebidas ou reservas via app → % da comissão vai para o FriendApp.
 - **API Comercial**
 - Endpoint: `/api/partners/payments`
 - Funções: registro de transações, cálculo de comissões, geração de relatórios.
-

◆ 3. Segurança e Confiabilidade dos Pagamentos

- **Gateway Integrado (Stripe/Adyen).**
 - **Carteira Digital FriendApp Wallet** → guarda FriendCoins e saldo em R\$.
 - **KYC/AML** obrigatório para Parceiros Visionários.
 - Logs financeiros criptografados (AES-256 em repouso + TLS em trânsito).
-

◆ 4. Métricas-Chave (KPIs para Investidores)

- ARPU (Average Revenue Per User).
 - LTV (Lifetime Value).
 - Take Rate em transações de parceiros.
 - Taxa de Adoção Premium (% usuários que convertem para pago).
 - GMV (Gross Merchandise Volume) de transações no ecossistema.
-

Fechamento

O FriendApp cria um **duplo motor de receita**:

- **B2C**: assinaturas, FriendCoins, serviços premium.
- **B2B**: SaaS para locais, comissões e patrocínios.

Esse modelo garante **previsibilidade de receita (assinaturas)** e **escalabilidade (comissões e transações)**.

Com isso, o sistema não só conecta pessoas, mas também **monetiza cada camada da experiência real**.

CAMADA 09 — LÓGICA DE MATCH ENERGÉTICO (MODO VIAGEM + BORA)

Entrada

Até aqui definimos **quem pode criar** (verificação), **como monetiza** (B2C + B2B), mas falta o **motor central de experiência social**:

🔑 **o algoritmo que mede afinidade, harmonia e colisões energéticas entre usuários, viagens e grupos Bora.**

Sem essa lógica, todo o sistema vira apenas uma rede social genérica.

Parte Técnica

◆ 1. Representação Matemática da “Energia”

Cada usuário no FriendApp é representado por um vetor no espaço vibracional multidimensional:

$$U_i = (f_1, f_2, f_3, \dots, f_n)$$

$$U_i = (f_1, f_2, f_3, \dots, f_n)$$

- Onde cada componente **f_k** é uma frequência latente (capturada pelo **perfil-e-frequência-service** e refinada pela **Aurah Kosmos**).
- Normalizado para unidade ($|U| = 1$), permitindo comparações por **similaridade de cosseno**.

Eventos/grupos também possuem vetores equivalentes (**E_j**) derivados da **intenção coletiva**.

◆ 2. Similaridade de Intenção (Compatibilidade de Contexto)

Afinidade entre **intenção de viagem/grupo** e o vetor do usuário:

$$\text{Sintencao}(U_i, E_j) = \cos(\theta) = \frac{U_i \cdot E_j}{|U_i| \cdot |E_j|}$$

$$\text{Sintencao}(U_i, E_j) = \cos(\theta) = \frac{U_i \cdot E_j}{|U_i| \cdot |E_j|}$$

Resultado $\in [0,1]$.

- $\geq 0.8 \rightarrow$ alta compatibilidade.
 - $0.5-0.79 \rightarrow$ média compatibilidade.
 - $< 0.5 \rightarrow$ baixa compatibilidade.
-

◆ 3. Score de Harmonia do Grupo Bora

Média da compatibilidade par-a-par entre todos os membros do grupo:

$$H = \frac{2}{n(n-1)} \sum_{i < j} \text{Sintencao}(U_i, U_j)$$

$$H = \frac{2}{n(n-1)} \sum_{i < j} \text{Sintencao}(U_i, U_j)$$

- Penalização: se a variância dos scores > 0.15 , aplica-se redução de 10% no score final (grupos heterogêneos demais).

◆ 4. Colisão Energética no Modo Viagem

Quando dois viajantes projetam sua energia para a mesma cidade e período, calcula-se:

$$C_{ij} = (0.6 \times \text{Sintencao}(U_i, U_j)) + (0.4 \times \text{Toverlap}) C_{\{ij\}} = (0.6 \times S_{\{\text{intencao}\}}(U_i, U_j)) + (0.4 \times T_{\{\text{overlap}\}})$$

$$C_{ij} = (0.6 \times \text{Sintencao}(U_i, U_j)) + (0.4 \times \text{Toverlap})$$

- **T_{overlap}** = proporção de sobreposição de datas (ex: 3 dias coincidentes em um período de 5 = 0.6).
 - Threshold: $C \geq 0.7$ gera notificação de colisão vibracional.
-

◆ 5. Score Final de Match

Para decidir recomendações (entrar em grupo Bora, sugerir conexão em viagem):

$$\text{Score}_{\text{final}} = (0.5 \times \text{Sintencao}) + (0.3 \times H) + (0.2 \times C) = (0.5 \times S_{\{\text{intencao}\}}) + (0.3 \times H) + (0.2 \times C)$$

$$\text{Score}_{\text{final}} = (0.5 \times \text{Sintencao}) + (0.3 \times H) + (0.2 \times C)$$

- Se > 0.75 → sugestão automática.
 - $0.6-0.74$ → sugestão opcional ("talvez seja interessante").
 - < 0.6 → nenhuma sugestão.
-

◆ 6. Exposição ao Usuário (UX Simples)

- O usuário nunca vê a fórmula → apenas representações amigáveis:
 - "Alta Harmonia" (Score ≥ 0.8).
 - "Boa Sincronia" ($0.6-0.79$).
 - "Pouca Conexão" (< 0.6).
-

◆ 7. APIs e Serviços Envolvidos

- `/api/match/similarity` → retorna S_{intencao} .
- `/api/match/group_harmony` → retorna H para um grupo Bora.
- `/api/match/collision` → retorna colisões previstas no Modo Viagem.

- **Aurah Kosmos Core:** centraliza vetores vibracionais.
 - **perfil-e-frequencia-service:** input inicial dos vetores.
-

Fechamento

Essa camada traduz a “**magia vibracional**” em **modelos matemáticos claros**: vetores, similaridade de cosseno, médias ponderadas.

- Para devs: dá clareza sobre **como implementar**.
- Para investidores: mostra que a experiência social é baseada em **cálculo científico** e não em promessas vagas.

CAMADA 10 — Governança e Segurança em Tempo Real (Botão de Alerta + Zonas Seguras + Mediação)

Entrada

A partir do momento em que o FriendApp começa a gerar encontros físicos por meio do **Modo Viagem** e dos grupos **Bora**, a **segurança do usuário** deixa de ser apenas vibracional e reputacional. Passa a ser também **física e em tempo real**.

Não basta oferecer um ecossistema confiável: o sistema precisa reagir imediatamente quando algo dá errado.

É aqui que entram as **mecanismos ativos de governança e segurança em tempo real**, que transformam o FriendApp em um ambiente protegido, prevenindo riscos e oferecendo respostas imediatas a incidentes.

Estrutura Técnica

Botão de Alerta Vibracional

- Disponível na tela do grupo Bora ou evento ativo.
- **Fluxo de acionamento:**
 1. Usuário pressiona o botão.

2. App dispara uma chamada para `/api/security/alert` .

3. Payload:

```
{
  "user_id": "u123",
  "group_id": "bora_789",
  "geo": { "lat": -23.562, "lng": -46.655 },
  "timestamp": "2025-08-19T22:00:00Z",
  "context": "live_meetup"
}
```

4. O backend registra o alerta em **logs criptografados** e dispara:

- Notificação silenciosa para a equipe de moderação central.
 - Push imediato para o criador do grupo.
 - SMS opcional para contato de emergência cadastrado.
- SLA: < **5 segundos** entre acionamento e notificação entregue.

Zonas de Encontro Seguras

- Integração direta com **Locais Parceiros Verificados**.
- IA Aurah Kosmos sugere (ou bonifica com XP/FriendCoins) encontros em locais certificados.
- Endpoints:
 - `/api/partners/secure_zones/list` — retorna todos os locais seguros próximos ao raio de 5 km.
 - `/api/meetups/create` exige flag `secure_zone=true` para grupos iniciais.

Sistema de Mediação em Tempo Real

- Camada de moderação proativa:
 - `/api/security/reports/live` → quando alertas múltiplos ocorrem no mesmo grupo.
 - Dispara **mediação ao vivo** via chat seguro, onde um moderador pode interagir com os usuários do grupo em tempo real.
- Políticas:

- **Silenciamento temporário** de membros em chat de grupo.
- **Banimento imediato** em caso de abuso comprovado.
- Logs armazenados no **Neo4j** para rastrear histórico de comportamento.

Fechamento

Com o **Botão de Alerta**, as **Zonas de Encontro Seguras** e a **Mediação em Tempo Real**, o FriendApp garante que encontros não sejam apenas vibracionalmente promissores, mas também fisicamente seguros.

CAMADA 11 — Controles de Privacidade e Visibilidade Granulares no Modo Viagem

Entrada

No **Modo Viagem**, o usuário projeta sua intenção de estar em outro lugar no futuro. Isso é poderoso, mas também **delicado**: compartilhar planos de viagem pode gerar riscos físicos (exposição pública de localização e datas).

Por isso, o FriendApp implementa **controles de privacidade granulares**, permitindo que cada usuário defina **quem pode ver sua projeção energética**.

Assim, a funcionalidade mantém sua **magia vibracional**, mas sem abrir mão da **segurança** e do **controle individual**.

Estrutura Técnica

Níveis de Visibilidade

No momento da ativação do Modo Viagem, o usuário escolhe um dos 4 níveis:

1. Apenas IA (Padrão)

- A projeção energética é invisível para qualquer humano.
- Só a IA Aurah Kosmos usa esses dados para sugerir colisões energéticas.

2. Apenas Conexões Autênticas

- Apenas amigos confirmados podem ver os planos.
- Endpoint:


```
GET /api/travel/projection?visibility=connections
```

3. Apenas Verificados Premium

- Torna os planos visíveis para um círculo maior, mas restrito a usuários verificados com selo Premium.
- Ideal para networking seguro.

4. Público

- Visível para todos na cidade de destino.
- Exige **consentimento extra** com aviso legal:

"Aviso: Tornar seus planos públicos aumenta a chance de conexões, mas também de exposição."

Abstração de Datas e Locais

- Para visualização pública, o sistema **abstrai detalhes sensíveis**:
 - Não mostra hotel específico.
 - Datas aparecem como **períodos aproximados**: "meados de agosto".
- Endpoint retorna payload simplificado:

```
{
  "user_id": "u456",
  "city": "Lisboa",
  "period": "Agosto (meados)",
  "visibility": "public"
}
```

Camada de Segurança Extra

- Logs de acesso: todo request `/api/travel/view` gera um log criptografado indicando quem visualizou a projeção.
- Sistema de notificações: usuário pode ativar alerta "quando alguém visualizar minha projeção pública".

← END Fechamento

Com os **controles de visibilidade granulares** e a **abstração de dados sensíveis**, o Modo Viagem se torna **seguro, confiável e personalizável**.

O usuário mantém total soberania sobre seus planos, equilibrando **privacidade** com **oportunidade de conexões reais**.

CAMADA 12 — Algoritmos Matemáticos de Score de Harmonia e Colisão Energética (Bora + Viagem)

Entrada

O coração da IA **Aurah Kosmos** é transformar estados vibracionais em **métricas quantificáveis**.

Aqui definimos as fórmulas que calculam:

1. **Score de Harmonia** (para grupos Bora).
2. **Score de Colisão Energética** (para encontros no Modo Viagem).

Essas métricas substituem abstrações como “alta sintonia” por **valores numéricos objetivos**, possibilitando:

- Ordenação de sugestões.
- Filtros em tempo real.
- Transparência para os usuários.

Estrutura Técnica

1. Representação Matemática da Aura

Cada usuário possui um vetor de estado vibracional **V_u** no espaço **n-dimensional**:

$$V_u = [f_1, f_2, f_3, \dots, f_n]$$

Onde cada componente representa **frequências normalizadas** (emoção, intenção, energia social, etc).

Exemplo simplificado (n=3):

- Emoção: 0.8
- Intenção: 0.6
- Energia Social: 0.9

$$V_u = [0.8, 0.6, 0.9]$$

2. Score de Harmonia (Grupos Bora)

A harmonia do grupo é baseada na **semelhança entre todos os vetores dos membros**.

Passo A — Similaridade de Pares

Entre dois usuários A e B, usamos **similaridade de cosseno**:

$$\text{sim}(A,B) = (V_a \cdot V_b) / (||V_a|| * ||V_b||)$$

Passo B — Média Geral

Para N usuários:

$$\text{Harmonia} = (\sum \text{sim}(i,j)) / (N*(N-1)/2)$$

Passo C — Penalização por Heterogeneidade

Se a variância dos vetores for alta, reduzimos o score:

$$\text{Harmonia}_{\text{final}} = \text{Harmonia} * (1 - \sigma^2)$$

Onde σ^2 = variância das magnitudes.

Escala final:

- 0.85 → Alta Harmonia.
- 0.65–0.85 → Harmonia Mediana.
- <0.65 → Harmonia Baixa.

3. Score de Colisão Energética (Modo Viagem)

Entre o viajante V_t e um residente V_r :

Fórmula Base:

Colisão = (α * SimilaridadeVetorial + β * CompatibilidadeDeIntenção + γ * HistóricoSocial)

- α = 0.5 (peso maior para vetores energéticos).
- β = 0.3 (peso médio para compatibilidade de intenção).
- γ = 0.2 (peso menor, mas importante, para histórico social positivo).

Compatibilidade de Intenção

Matriz pré-definida com valores entre 0 e 1:

Exemplo:

- Cura ↔ Espiritualidade = 0.9
- Festa ↔ Cura = 0.2

Histórico Social

Se já houve encontros positivos, adiciona bônus (até +0.15).

Escala final:

- 0.80 → Colisão Forte (recomendação destacada).
- 0.60–0.80 → Colisão Potencial.
- <0.60 → Colisão Fraca (não exibida).

4. Implementação API

- **Endpoint Harmonia (Grupo Bora):**

GET /api/bora/harmony-score/{group_id}

Resposta:

```
{
  "group_id": "g123",
  "harmonia_score": 0.82,
  "nivel": "Harmonia Mediana"
}
```

- **Endpoint Colisão (Modo Viagem):**

```
GET /api/travel/collision-score?traveler={id}&resident={id}
```

Resposta:

```
{
  "traveler_id": "u123",
  "resident_id": "u456",
  "colisao_score": 0.87,
  "nivel": "Colisão Forte"
}
```

Fechamento

Com essas fórmulas, a IA Aurah Kosmos ganha **transparência operacional**:

- Desenvolvedores podem implementar sem ambiguidade.
- Investidores entendem a solidez matemática.
- Usuários recebem recomendações claras ("Alta Harmonia", "Colisão Forte").

CAMADA 13 — PROTOCOLO DE SEGURANÇA EM TEMPO REAL (BORA +

VIAGEM)

Entrada

O FriendApp se compromete a ser **mais que uma rede social vibracional**: ele é também um **ambiente seguro** para encontros no mundo real. Como o módulo **Bora + Modo Viagem** incentiva interações presenciais, é indispensável que a segurança não seja apenas reativa (denúncias posteriores), mas **proativa e em tempo real**.

Esta camada estabelece um **protocolo robusto de segurança**, integrando botões de emergência, contatos de confiança, IA de monitoramento e incentivo a encontros em **Zonas Seguras (Locais Parceiros)**.

Núcleo Técnico

Botão de Alerta Vibracional

- Disponível em encontros Bora ativos e check-ins do Modo Viagem.
- Ícone discreto mas fixo na tela.
- Ações ao ser acionado:
 - Envia sinal **silencioso e imediato** para o **time de moderação FriendApp**.
 - Notifica o **criador do grupo Bora** (se aplicável).
 - Oferece ao usuário opções:
 - Compartilhar **localização em tempo real** com contato de emergência.
 - Atalho direto para **números locais de emergência**.
 - Encerrar participação no evento.

```
POST /api/security/alert
{
  "user_id": "USR123",
  "group_id": "BOR456",
  "location": { "lat": -23.5617, "lng": -46.6559 },
  "share_with_guardian": true,
  "timestamp": "2025-08-19T21:14:00Z"
```

```
}
```

Zonas de Encontro Seguras (Locais Parceiros)

- **Primeiros encontros Bora** são **fortemente recomendados** (gamificação: XP + FriendCoins) em **parceiros verificados**.
- Benefícios:
 - Local público + infraestrutura básica (iluminação, wi-fi, vigilância).
 - Parceiros recebem destaque no app e métricas de retenção.
- API de recomendação conecta Bora → Locais Parceiros via IA Aurah Kosmos.

```
GET /api/partners/safe-zones?city=São Paulo
[
  { "id": "LOC123", "name": "Café Harmonia", "verified": true, "score": 9.1 }
]
```

Contatos de Confiança (Guardianship)

- Usuário define até **3 Guardiões de Segurança** nas configurações.
- Em encontros Bora ou Viagem:
 - Pode compartilhar **rota e check-in** com 1 clique.
 - Guardião recebe notificação push + SMS com link de rastreamento temporário.

```
POST /api/security/share-location
{
  "user_id": "USR123",
  "guardian_id": "GUARD567",
  "session": "BOR456",
  "expires_in": 3600
}
```

IA de Monitoramento Preventivo

- **Aurah Kosmos** calcula risco de encontros com base em:
 - Local (horário, histórico de incidentes).
 - Score de harmonia do grupo.
 - Reputação vibracional dos participantes.

- Se risco alto → alerta preventivo:

"Aurah percebe energia instável neste encontro. Recomendamos optar por um Local Parceiro."

Fluxo de Governança Técnica

1. **Usuário dispara alerta → backend registra log criptografado.**
 2. **Notificação imediata → time de moderação + guardião.**
 3. **Fallback:** se sem resposta, escalonar para autoridades locais (via APIs 3rd-party, onde legalmente permitido).
 4. **Encerramento:** logs armazenados no **banco de eventos críticos** (PostgreSQL com criptografia AES256).
-

Fechamento

Com o **Protocolo de Segurança em Tempo Real**, o FriendApp elimina a fragilidade dos encontros sociais espontâneos.

Não se trata apenas de **vibração e sincronicidade**, mas de um sistema **matematicamente seguro, auditável e preventivo**.

CAMADA 14 — PRIVACIDADE GRANULAR E CONTROLES DE VISIBILIDADE (MODO VIAGEM + BORA)

Entrada


Um dos maiores riscos de funcionalidades que lidam com **viagens futuras** e **encontros presenciais** é a exposição excessiva das informações pessoais do usuário. Para evitar vulnerabilidades, esta camada estabelece **políticas técnicas e arquitetônicas** de privacidade granular, onde o usuário controla exatamente *quem vê o quê e quando*.

Estrutura Técnica Detalhada

1. Níveis de Visibilidade (Configuração pelo Usuário)

Cada vez que o **Modo Viagem** ou um **Bora** é criado, o usuário define quem poderá visualizar suas intenções de conexão:

- **Privado (IA apenas)** → Projeção energética invisível para outros, mas usada internamente pela IA Aurah Kosmos.
- **Conexões Autênticas** → Apenas amigos adicionados podem ver a intenção.
- **Usuários Verificados Premium** → Rede mais ampla de confiança (usuários com selo DUC/DCO + Premium).
- **Público** → Todos os usuários podem visualizar (com aviso explícito de riscos).

 **Armazenamento:** Configuração de visibilidade persistida no `user_privacy_prefs` no banco de dados relacional (PostgreSQL).

2. Abstração de Dados Sensíveis

- **Localização** → Nunca exibir coordenadas exatas em modo público. Apenas áreas amplas (ex: "Vila Madalena, São Paulo").
 - **Datas** → Arredondar intervalos para períodos ("meados de agosto") em vez de datas exatas, exceto quando já há conexão confirmada.
 - **Horários** → Só exibidos após match mútuo.
-

3. Políticas de Anonimização

- Logs de viagem e grupos Bora armazenados com **hashing irreversível de IDs**.
 - Dados de eventos apagados ou anonimizados após **90 dias** (LGPD compliance).
 - Implementação de **chaves rotativas de criptografia (AES-256)** em repouso.
-

4. Governança de Acesso

- **RBAC (Role-Based Access Control)** aplicado em todos os endpoints.
- Somente **donos do grupo Bora** podem ver todos os detalhes dos membros (dentro do grupo).

- Moderadores têm acesso apenas aos metadados necessários para mediação de conflitos.

5. APIs Técnicas

```
# Definir nível de visibilidade
POST /api/privacy/set_visibility
{
  "feature": "modo_viagem",
  "visibility_level": "connections_only"
}

# Obter configuração atual
GET /api/privacy/get_visibility?feature=bora

# Exemplo de resposta
{
  "feature": "bora",
  "visibility_level": "premium_verified",
  "expires_at": "2025-09-01T23:59:59Z"
}
```

← END Fechamento

Com a privacidade granular implementada, o FriendApp garante que o **usuário mantém o controle total** sobre quem pode ver seus movimentos energéticos e presenciais. Essa camada reduz riscos de exposição excessiva e aumenta a confiança no uso do **Modo Viagem** e dos grupos **Bora**, assegurando conformidade com normas como **LGPD/GDPR** e boas práticas de segurança.

CAMADA 15 — FÓRMULAS MATEMÁTICAS DO SCORE DE HARMONIA E COLISÃO ENERGÉTICA (IA AURAH KOSMOS)

Entrada

Até aqui, falamos sobre “match vibracional” e “colisões energéticas” de forma conceitual. Mas para a engenharia, precisamos de **modelos matemáticos explícitos**, que possam ser **implementados, testados e calibrados**.

Esta camada define as **equações base** que a IA Aurah Kosmos utiliza para calcular:

1. **Score de Harmonia** em grupos Bora.
2. **Colisão Energética** no Modo Viagem.
3. **Afinidade de Intenção** entre usuários e eventos/locais.

Estrutura Técnica Detalhada

1. Representação Vetorial da Energia

Cada usuário possui um vetor vibracional V_u (dimensão d , tipicamente $d=128$), gerado a partir de:

- Estado emocional (NLP + sensoriamento vibracional).
- Frequência atual ($f_{frequencia_atual}$).
- Histórico de interações positivas/negativas.
- Intenções declaradas.

Exemplo:

$$V_u = [0.12, -0.34, 0.87, \dots, 0.21]$$

Todos os cálculos seguintes usam essas representações.

2. Score de Harmonia (Grupos Bora)

A harmonia de um grupo é definida pela **média de similaridade de cosseno** entre todos os pares de usuários:

$$H = \frac{2}{n(n-1)} \sum_{i=1}^n \sum_{j=i+1}^n \text{sim}(V_{u_i}, V_{u_j})$$


$$H = \frac{1}{n(n-1)} \sum_{i=1}^n \sum_{j=i+1}^n \text{sim}(V_{u_i}, V_{u_j})$$

onde:

$$\text{sim}(V_{u_i}, V_{u_j}) = \frac{V_{u_i} \cdot V_{u_j}}{\|V_{u_i}\| \cdot \|V_{u_j}\|} \quad \text{sim}(V_{\{u_i\}}, V_{\{u_j\}}) = \frac{V_{\{u_i\}} \cdot V_{\{u_j\}}}{\|V_{\{u_i\}}\| \cdot \|V_{\{u_j\}}\|}$$

$$\text{sim}(V_{u_i}, V_{u_j}) = \frac{V_{u_i} \cdot V_{u_j}}{\|V_{u_i}\| \cdot \|V_{u_j}\|}$$

- **Intervalo:** $0 \leq H \leq 1$
- **Interpretação:**
 - $H > 0.8$ → Grupo muito harmônico.
 - $0.6 \leq H \leq 0.8$ → Harmonia média.
 - $H < 0.6$ → Grupo heterogêneo, risco de fricção.

 **Implementação:** cálculo feito em batch no serviço `conexoes-reais-service`, armazenado em `group_metrics`.

3. Colisão Energética (Modo Viagem)

Para prever encontros no Modo Viagem, calculamos a **colisão energética** entre viajante `u` e residente `r`:

$$C(u, r) = \alpha \cdot \text{sim}(V_u, V_r) + \beta \cdot f(\text{intencao}_u, \text{intencao}_r) \quad C(u, r) = \alpha \cdot \text{sim}(V_u, V_r) + \beta \cdot f(\text{intencao}_u, \text{intencao}_r)$$

$$C(u, r) = \alpha \cdot \text{sim}(V_u, V_r) + \beta \cdot f(\text{intencao}_u, \text{intencao}_r)$$

onde:

- `sim` = similaridade de cosseno entre vetores.
- `f(intencao)` = compatibilidade de intenções (matriz pré-definida, escala 0–1).
- α, β = pesos ajustáveis (default: `0.5, 0.5`).

 **Match confirmado se:** $C(u, r) \geq 0.75$.

4. Afinidade Usuário–Evento/Local

Para recomendar eventos ou locais:

$$A(u, e) = \gamma \cdot \text{sim}(V_u, V_e) + \delta \cdot h(u, e) + \epsilon \cdot g(u, e) \quad A(u, e) = \gamma \cdot \text{sim}(V_u, V_e) + \delta \cdot h(u, e) + \epsilon \cdot g(u, e)$$

$$A(u, e) = \gamma \cdot \text{sim}(V_u, V_e) + \delta \cdot h(u, e) + \epsilon \cdot g(u, e)$$

onde:

- `sim(V_u, V_e)` → compatibilidade vibracional.
- `h(u, e)` → histórico do usuário com eventos/locais similares (0–1).

- $g(u, e)$ → peso social (amigos participando).
- Pesos padrão: $\gamma=0.6, \delta=0.2, \varepsilon=0.2$.

 **Exibição no app:** eventos/locais com $A(u, e) \geq 0.7$.

5. APIs Técnicas

```
# Calcular score de harmonia de grupo
GET /api/ia/harmonia?group_id=123

# Exemplo de resposta
{
  "group_id": "123",
  "harmonia_score": 0.82,
  "nivel": "Alta Harmonia"
}

# Calcular colisão energética entre viajante e residente
POST /api/ia/colisao
{
  "user_a": "u123",
  "user_b": "u987"
}

# Resposta
{
  "score_colisao": 0.78,
  "match": true
}
```

Fechamento

Com estas fórmulas explícitas, o conceito de **harmonia vibracional** deixa de ser místico e passa a ser **matematicamente implementável**. Isso permite:

- Testes unitários claros para os devs.
- Ajustes calibrados de pesos ($\alpha, \beta, \gamma, \delta, \varepsilon$).

- Transparência para investidores (mostramos que é engenharia, não “magia”).

CAMADA 16 — GOVERNANÇA DE GRUPOS BORA (MODERAÇÃO, DENÚNCIAS E SLA DE VERIFICAÇÃO)

Entrada

O **Bora** é o coração da experiência de encontros espontâneos no FriendApp. Mas, ao permitir que qualquer usuário verificado crie grupos, abrimos espaço para **dinâmicas sociais complexas**: conflitos, abusos de poder, ou má utilização da ferramenta.


Esta camada define as **regras de governança**, os **mecanismos técnicos de moderação** e o **SLA do processo de verificação (DUC/DCO)** para garantir que os encontros presenciais aconteçam de forma **segura, justa e sustentável**.

Estrutura Técnica Detalhada

1. Papéis e Permissões no Bora

Cada grupo Bora terá três níveis de papéis:

- **Criador (Host):**
 - Pode convidar, aprovar ou remover membros.
 - Pode **silenciar** um usuário por até 24h (não apenas expulsar).
 - Pode encerrar o grupo a qualquer momento.
- **Moderador (opcional):**
 - Pode aprovar solicitações de entrada.
 - Pode silenciar membros temporariamente.
 - **Não pode expulsar** nem encerrar o grupo.
- **Membros:**
 - Podem interagir, postar mensagens, reagir.
 - Podem denunciar usuários, mensagens ou o próprio criador.

 **Controle técnico:** essas permissões ficam no microserviço `conexoes-reais-service`, com RBAC (Role-Based Access Control).

2. Sistema de Denúncias

Denúncias podem ser feitas em três níveis:

- **Usuário específico no Bora** → denúncia pessoal (assédio, spam, abuso).
- **Mensagem/post dentro do Bora** → denúncia de conteúdo (ódio, nudez, golpe).
- **Grupo inteiro** → denúncia de má conduta do criador ou uso indevido do recurso.

 **Fluxo da denúncia:**

1. Usuário seleciona motivo → abre ticket no `suporte-e-comunidade-service`.
 2. Ticket é classificado automaticamente via **IA de Moderação** (prioridade alta/baixa).
 3. Moderadores humanos atuam em casos críticos.
 4. Penalidades automáticas:
 - Bloqueio temporário de criação de novos grupos.
 - Redução no **Score de Reputação** do usuário.
 - Banimento em caso de reincidência.
-

3. SLA do Processo de Verificação (DUC/DCO)

Para criar grupos Bora, o usuário precisa passar pela **verificação documental**. Isso gera o risco de **atrito** se a validação for lenta.

 **Parâmetros de SLA:**

- **Verificações automáticas (OCR + IA):**
 - 85% dos casos → resultado em **menos de 2 minutos**.
- **Casos manuais (revisão humana):**
 - 95% resolvidos em até **5 minutos**.
 - 100% resolvidos em até **15 minutos**.

 **Gamificação da verificação proativa:**

Usuários são incentivados a se verificar **antes** de precisar criar um grupo.

Missão no Jogo da Transmutação:

“🌐 Ancore sua Identidade Real — conclua a verificação e receba +100 XP e +20 FriendCoins”.

4. Logs e Auditoria

- Cada ação dentro do Bora (expulsar, silenciar, denúncia) gera um **log criptografado**.
- Logs ficam armazenados em **PostgreSQL** + backup em **Firestore** para consultas rápidas.
- Auditorias podem ser solicitadas em caso de disputas (ex: membro alegando expulsão injusta).

5. APIs Técnicas

```
# Denunciar membro em um Bora
```

```
POST /api/bora/denuncia
```

```
{  
  "group_id": "bora_456",  
  "target_user": "user_789",  
  "motivo": "assédio"  
}
```

```
# Resposta
```

```
{  
  "status": "aberto",  
  "ticket_id": "dnc_1023",  
  "prioridade": "alta"  
}
```

```
# Silenciar usuário por 24h
```

```
POST /api/bora/silenciar
```

```
{  
  "group_id": "bora_456",  
  "target_user": "user_789",  
  "tempo_horas": 24  
}
```



```
}

# Resposta
{
  "status": "ok",
  "expira_em": "2025-08-20T12:00:00Z"
}
```

Fechamento

Com esta governança, o Bora deixa de ser apenas um **chat espontâneo** e passa a ser um espaço **seguro, regulado e auditável**, sustentado por:

- Papéis claros (Host, Moderador, Membro).
- Denúncias multilayer com resposta rápida.
- SLA forte para verificação documental.
- Auditoria garantida com logs criptografados.

CAMADA 17 — POLÍTICAS DE PRIVACIDADE E CONTROLES DE VISIBILIDADE (MODO VIAGEM)

Entrada

Um dos maiores riscos do **Modo Viagem** é expor informações sensíveis do usuário, como **planos futuros de deslocamento**, locais exatos ou datas específicas. Para evitar vulnerabilidades físicas e digitais, o sistema precisa fornecer **camadas granulares de privacidade**, dando ao usuário controle total sobre sua projeção energética.

Definições e Níveis de Visibilidade

O sistema define **quatro níveis de visibilidade** configuráveis no momento de ativação do Modo Viagem:

1. Somente IA (padrão)

- A projeção energética é **invisível para outros usuários**.

- Apenas a **Aurah Kosmos** utiliza os dados para calcular **colisões energéticas preditivas**.

- Nenhum dado é exibido publicamente até que o usuário permita.

2. Conexões Autênticas

- Apenas amigos confirmados podem visualizar planos de viagem.
- Ideal para coordenação entre pessoas que já possuem laços confiáveis.

3. Usuários Premium Verificados

- Disponível apenas para membros Premium com **DUC/DCO ativo**.
- Permite ampliar o alcance da projeção sem comprometer segurança.

4. Público Geral (alto risco)

- Todos os usuários da cidade de destino podem ver a projeção.
- O sistema exige **confirmação extra** e exibe avisos de risco antes da ativação.

Abstração de Dados (Proteção Adicional)

- **Localização:** em modo público, nunca mostrar endereços exatos ou hotéis, apenas **regiões amplas** ("Centro", "Vila Madalena", "Zona Sul").
- **Datas:** exibir **janelas temporais** ("meados de agosto") em vez de datas exatas.
- **Precisão dinâmica:** detalhes só são revelados quando houver uma **colisão energética mútua** validada.

Lógica Técnica

- **Tabela de Configuração de Privacidade** armazenada no `perfil-e-frequencia-service`:

```
{
  "user_id": "12345",
  "modo_viagem": {
    "destino": "Lisboa",
    "data_inicio": "2025-08-20",
    "data_fim": "2025-08-28",
    "visibilidade": "autenticados",
```

```
"nivel_abstracao": {  
  "local": "bairro",  
  "data": "janela_temporal"  
}  
}  
}
```

- **APIs Principais:**

- `POST /api/modo-viagem/configurar` → cria ou altera projeção.
- `GET /api/modo-viagem/visualizar` → retorna projeções visíveis de acordo com permissões.
- `GET /api/modo-viagem/privacidade` → consulta os níveis de visibilidade aplicados.

- **Autorização:** todas as rotas exigem **JWT + escopo de permissão**, com checagem de roles (Premium, Verificado, Público).

Segurança

- **Criptografia em trânsito e em repouso** (TLS 1.3 + AES-256).
- **Rate limiting** para evitar scraping massivo de projeções.
- **Logs auditáveis** em caso de vazamento ou abuso.

Fechamento

Esta camada garante que o **Modo Viagem** seja uma ferramenta poderosa **sem abrir vulnerabilidades pessoais**.

O usuário escolhe como e quando se expor, enquanto a IA Aurah Kosmos continua operando silenciosamente para **maximizar encontros significativos**.

CAMADA 18 — ALGORITMOS MATEMÁTICOS DE COMPATIBILIDADE (SCORE DE HARMONIA + COLISÃO ENERGÉTICA)

Entrada

Até aqui, definimos como os usuários projetam energia (Modo Viagem) e se conectam em grupos espontâneos (Bora).

Agora, precisamos **matematizar** a compatibilidade vibracional, eliminando qualquer ambiguidade para o time técnico.

Essa camada traz as **fórmulas vetoriais, ponderações e métricas objetivas** que regem a lógica de matching energético.

1. Representação Vetorial da Energia

Cada usuário e cada grupo/evento é representado como um **vetor de frequência energética** em um espaço multidimensional:

- **Dimensões principais (normalizadas entre 0 e 1):**
 - Intenção (cura, diversão, expansão, conexão, aprendizado).
 - Estado emocional atual (leve, introspectivo, expansivo, vibrante).
 - Frequência de interação (ativa, média, baixa).
 - Score de reputação (calculado pela IA de confiança).

👉 Exemplo de vetor de usuário:

$U = [0.9 \text{ (cura)}, 0.7 \text{ (expansivo)}, 0.4 \text{ (interac,a\~o)}, 0.85 \text{ (reputac,a\~o)}]$
 $U = [0.9 \setminus (\text{cura}), \setminus 0.7 \setminus (\text{expansivo}), \setminus 0.4 \setminus (\text{interac\~ao}), \setminus 0.85 \setminus (\text{reputac\~ao})]$

$U = [0.9 \text{ (cura)}, 0.7 \text{ (expansivo)}, 0.4 \text{ (interac,a\~o)}, 0.85 \text{ (reputac,a\~o)}]$

2. Score de Harmonia (Grupos Bora)

O **Score de Harmonia** mede a coerência interna de um grupo Bora:

$$H = \frac{1}{N} \sum_{i=1}^N \sum_{j=i+1}^N \text{sim}(U_i, U_j) - \text{var}(S)$$
$$H = \frac{1}{N} \sum_{i=1}^N \sum_{j=i+1}^N \text{sim}(U_i, U_j) - \text{var}(S)$$

$$H = \frac{1}{N} \sum_{i=1}^N \sum_{j=i+1}^N \text{sim}(U_i, U_j) - \text{var}(S)$$

- **sim(U_i, U_j)** = Similaridade de cossenos entre vetores de usuários.
- **var(S)** = variância da distribuição de scores individuais do grupo (quanto mais homogêneo, maior a harmonia).
- Escala final: **0 a 1**, convertida para **0 a 100%** no app.

👉 Exemplo:

- Grupo de 5 pessoas, média de similaridade = 0.82.

- Variância = 0.05.
- Harmonia = 0.77 (77%) → exibido como "Alta Harmonia".

3. Colisão Energética (Modo Viagem)

A **Colisão Energética** prevê se dois usuários devem se encontrar fisicamente em uma cidade de destino.

$$C = (0.5 \cdot \text{sim}(U, V)) + (0.3 \cdot \text{int}(U, V)) + (0.2 \cdot \text{geo}(U, V))$$

$$C = (0.5 \cdot \text{sim}(U, V)) + (0.3 \cdot \text{int}(U, V)) + (0.2 \cdot \text{geo}(U, V))$$

- **sim(U, V)** = Similaridade de cossenos dos vetores energéticos.
- **int(U, V)** = Compatibilidade entre intenções (tabela de afinidade pré-definida).
- **geo(U, V)** = Proximidade geográfica (quanto menor a distância, maior o score).

👉 Regra prática:

- Se **C ≥ 0.75**, a IA gera **notificação preditiva de encontro**.
- Se **0.60 ≤ C < 0.75**, a IA recomenda apenas de forma passiva ("pessoas com energia próxima também estarão em Lisboa").

4. Afinidade com Locais Parceiros

Para cruzar usuários com locais parceiros (cafés, centros culturais, etc.):

$$L = (0.6 \cdot \text{tag_match}(U, P)) + (0.4 \cdot \text{geo}(U, P))$$

$$L = (0.6 \cdot \text{tag_match}(U, P)) + (0.4 \cdot \text{geo}(U, P))$$

- **tag_match(U, P)** = Similaridade entre intenção do usuário e categorias vibracionais do local (ex: "cura" combina com "centro holístico").
- **geo(U, P)** = Distância física até o local.

5. Tabela de Thresholds

Score (0-1)	Interpretação	Ação no App
≥ 0.85	Conexão Rara	Notificação imediata ("Encontro altamente vibracional previsto")

Score (0-1)	Interpretação	Ação no App
0.75–0.84	Conexão Forte	Sugestão direta de match/entrada em grupo
0.60–0.74	Conexão Média	Sugestão passiva, exibida no feed energético
< 0.60	Conexão Fraca	Não exibe notificações

APIs e Implementação

- `POST /api/match/calculate` → recebe dois vetores e retorna score de colisão.
- `POST /api/group/harmony` → recebe lista de usuários e retorna harmonia do grupo.
- `GET /api/partners/affinity` → retorna ranking de locais mais compatíveis.

Fechamento

Esta camada fecha a “caixa preta” dos cálculos vibracionais:

- Fórmulas matemáticas claras.
- Scores normalizados e thresholds objetivos.
- APIs definidas para uso direto pelos microserviços.

CAMADA 19 — LÓGICA DE AGENDA, SINCRONIA TEMPORAL E COMPATIBILIDADE DE DATAS (MODO VIAGEM + BORA + PARCEIROS)

Entrada

Esta camada transforma intenção em encontro **viável no tempo**. O objetivo é alinhar:

1. **janelas de disponibilidade** de participantes (inclui viajantes),
2. **restrições do local parceiro** (horário, capacidade, bloqueios),
3. **regras operacionais** (fuso horário, duração mínima, buffers, lead time),
para gerar **slots candidatos** ranqueados e confirmáveis com controle de concorrência.

Parte Técnica (99%)

1) Normalização temporal

- **Formato canônico:** `timestampz` (UTC) no banco; renderização em TZ local do destino/usuário.
- **TZ de referência por contexto:**
 - Modo Viagem → `tz_destino`.
 - Bora local → `tz_local_parceiro` (quando existir) ou `tz_criador`.
- **Granularidade:** slots discretizados em **15 min** (configurável).
- **Buffers:**
 - `buffer_troca_local` (ex.: 15 min) entre encontros consecutivos.
 - `buffer_deslocamento` estimado via API de mapa (opcional).
- **DST:** proibido agendar em intervalos “inexistentes”/“duplicados” (transição de horário de verão). Validação obrigatória.

2) Modelo de dados (PostgreSQL sugerido)

```
-- Janela de viagem declarada (modo viagem)
CREATE TABLE travel_itinerary (
  travel_id UUID PRIMARY KEY,
  user_id   UUID NOT NULL,
  tz_dest   TEXT NOT NULL,      -- IANA, ex: 'Europe/Lisbon'
  window    TSTZRANGE NOT NULL, -- [inicio_utc, fim_utc)
  intent    JSONB,              -- metadados de intenção
  created_at TIMESTAMPTZ DEFAULT NOW()
);
CREATE INDEX ON travel_itinerary USING GIST (window);

-- Disponibilidades recorrentes do usuário (por dia da semana, na TZ do usuário)
CREATE TABLE user_weekly_availability (
  user_id UUID NOT NULL,
  dow     INT NOT NULL,        -- 0=Dom .. 6=Sáb
  start_local TIME NOT NULL,
  end_local  TIME NOT NULL,
```

```

tz_user    TEXT NOT NULL,
PRIMARY KEY (user_id, dow)
);

-- Exceções pontuais (compromissos/bloqueios do calendário)
CREATE TABLE user_blackout (
  user_id UUID NOT NULL,
  window TSTZRANGE NOT NULL,
  reason TEXT,
  PRIMARY KEY (user_id, window)
);
CREATE INDEX ON user_blackout USING GIST (window);

-- Local parceiro (TZ e janelas operacionais)
CREATE TABLE partner_venue (
  venue_id UUID PRIMARY KEY,
  name TEXT,
  tz TEXT NOT NULL,
  min_duration_min INT DEFAULT 60
);

-- Horário de funcionamento semanal do parceiro
CREATE TABLE partner_open_hours (
  venue_id UUID NOT NULL,
  dow INT NOT NULL,
  open_local TIME NOT NULL,
  close_local TIME NOT NULL,
  PRIMARY KEY (venue_id, dow)
);

-- Fechamentos/bloqueios específicos (feriado, manutenção)
CREATE TABLE partner_blackout (
  venue_id UUID NOT NULL,
  window TSTZRANGE NOT NULL,
  reason TEXT,
  PRIMARY KEY (venue_id, window)
);
CREATE INDEX ON partner_blackout USING GIST (window);

```



```

-- Capacidade do parceiro (janelas e lotação)
CREATE TABLE partner_capacity (
  venue_id UUID NOT NULL,
  window  TSTZRANGE NOT NULL,
  capacity INT NOT NULL,
  holds   INT NOT NULL DEFAULT 0,
  bookings INT NOT NULL DEFAULT 0,
  PRIMARY KEY (venue_id, window)
);
CREATE INDEX ON partner_capacity USING GIST (window);

-- Grupo Bora
CREATE TABLE bora_group (
  group_id  UUID PRIMARY KEY,
  host_id   UUID NOT NULL,
  venue_id  UUID,                -- opcional (Bora sem parceiro definido)
  duration_min INT NOT NULL CHECK (duration_min > 0),
  tz_ref    TEXT NOT NULL,
  lead_time_min INT DEFAULT 60
);

-- Propostas de slots geradas
CREATE TABLE bora_proposed_slot (
  slot_id  UUID PRIMARY KEY,
  group_id UUID NOT NULL,
  start_utc TIMESTAMPTZ NOT NULL,
  end_utc  TIMESTAMPTZ NOT NULL,
  venue_id UUID,
  score    NUMERIC(5,4) NOT NULL,
  status   TEXT NOT NULL CHECK (status IN ('proposed','hold','expired','confirmed','rejected')),
  created_at TIMESTAMPTZ DEFAULT NOW()
);
CREATE INDEX ON bora_proposed_slot (group_id, start_utc);

```

Observações:

- Usar **tstzrange** + **GIST** para consultas por interseção é crítico para performance.

- `user_weekly_availability` é expandido para períodos concretos no cálculo de candidatos.

3) Regras de viabilidade de um slot (predicados)

Um **slot candidato** `[s,e]` é viável **SE E SOMENTE SE**:

1. Interseção de disponibilidade de todos os participantes:

$[s,e] \subseteq p \in \text{participantes} \cap \text{Disponibilidade_expandida}(p)$

$[s,e] \subseteq \bigcap_{p \in \text{participantes}} \text{Disponibilidade_expandida}(p)$ $[s,e] \setminus \text{subseq} \bigcap_{p \in \text{participantes}} \text{Disponibilidade_expandida}(p)$

(considerando TZs, buffers, e removendo `user_blackout`).

2. Janela de viagem (para viajantes): `[s,e] \subseteq window_viagem` (em UTC) para cada viajante do grupo.

3. Local parceiro (se definido):

- `[s,e]` dentro do horário operacional do parceiro (convertendo `open_local/close_local` para UTC).
- Sem interseção com `partner_blackout.window`.
- **Capacidade:** `bookings + holds + tamanho_grupo \leq capacity` no range correspondente.

4. Lead time: `s - now() \geq lead_time_min` (grupo e/ou parceiro).

5. Buffers:

- Distância temporal do último compromisso de cada participante \geq `buffer_troca_local + buffer_deslocamento`.

4) Algoritmo de geração de slots (visão geral)

Passo A — Expandir disponibilidades semanais para o horizonte de datas alvo (ex.: próximos 14 dias), convertendo **local** \rightarrow **UTC** e aplicando `user_blackout` e `travel_itinerary.window`.

Passo B — Interseção multiconjunto (varredura “line sweep”):

- Unir todos os intervalos por usuário;
- Intersectar conjuntos entre participantes;
- Cortar em janelas mínimas de `duration_min`.

Passo C — Aplicar restrições do parceiro (se houver): `open_hours`, `blackout`, `capacity`, `lead_time`.

Passo D — Calcular score de cada candidato e ranquear **Top-N**.

5) Função de score (ranqueamento de candidatos)

$$\text{Score}(s,e) = w_1 \cdot \text{Phora}(s) + w_2 \cdot L(s) + w_3 \cdot R_{\text{cap}}(s,e) + w_4 \cdot Q_{\text{venue}} + w_5 \cdot H_{\text{grupo}} \\ \text{Score}(s,e) = w_1 \cdot P_{\text{hora}}(s) + w_2 \cdot L(s) + w_3 \cdot R_{\text{cap}}(s,e) + w_4 \cdot Q_{\text{venue}} + w_5 \cdot H_{\text{grupo}}$$
$$\text{Score}(s,e) = w_1 \cdot \text{Phora}(s) + w_2 \cdot L(s) + w_3 \cdot R_{\text{cap}}(s,e) + w_4 \cdot Q_{\text{venue}} + w_5 \cdot H_{\text{grupo}}$$

- $\text{Phora}(s)$ $P_{\text{hora}}(s)$ $\text{Phora}(s)$ — preferência média de horário dos participantes (0–1) (ex.: manhã/tarde/noite).
- $L(s)$ $L(s)$ $L(s)$ — penalidade por urgência (respeito ao lead time):
$$L(s) = \min(1, s - \text{now} / \text{lead_time_alvo})$$
$$L(s) = \min(1, \frac{s - \text{now}}{\text{lead_time_alvo}})$$
- R_{cap} R_{cap} R_{cap} — “folga” de capacidade:
$$\max(0, 1 - \frac{\text{bookings} + \text{holds} + \text{tamanho_grupo}}{\text{capacity}})$$
$$\max(0, 1 - \frac{\text{bookings} + \text{holds} + \text{tamanho_grupo}}{\text{capacity}})$$
- Q_{venue} Q_{venue} Q_{venue} — qualidade/afinidade do local (0–1) (derivada de reputação/adequação).
- H_{grupo} H_{grupo} H_{grupo} — score de harmonia do grupo (0–1) (da Camada 12/15).

Pesos padrão: $w_1=0.25$, $w_2=0.15$, $w_3=0.25$, $w_4=0.15$, $w_5=0.20$
 $w_1=0.25$, $w_2=0.15$, $w_3=0.25$, $w_4=0.15$, $w_5=0.20$
 $w_1=0.25$, $w_2=0.15$, $w_3=0.25$, $w_4=0.15$, $w_5=0.20$.

(Pesos calibráveis via experimentos A/B.)

6) Pseudocódigo (geração de candidatos)

```
def generate_slots(group_id, participants, venue_id, duration_min, horizon_days=14):
    # 1) Coletar dados base
    tz_ref = get_group_tz(group_id)
    avail_maps = []
    for u in participants:
        weekly = get_user_weekly_availability(u)    # local times
        blackouts = get_user_blackouts(u)          # tstzrange
```

```

travel = get_travel_itineraries(u)          # optional
expanded = expand_to_utc(weekly, tz_user(u), horizon_days)
expanded = intersect_with_travel(expanded, travel)
expanded = subtract_blackouts(expanded, blackouts)
expanded = apply_buffers(expanded, u)
avail_maps.append(expanded)

# 2) Interseção de todos
common = intersect_all(avail_maps)          # list of [start_utc,end_utc)

# 3) Subdividir por duração
candidates = slice_by_duration(common, duration_min, step=15*MINUTES)

# 4) Filtrar por parceiro (se houver)
if venue_id:
    open_ranges = get_partner_open_utc(venue_id, horizon_days)
    p_blackouts = get_partner_blackouts(venue_id)
    candidates = intersect_with_ranges(candidates, open_ranges)
    candidates = subtract_ranges(candidates, p_blackouts)
    candidates = filter_by_capacity(candidates, venue_id, group_size= len(p
    participants))
    candidates = filter_by_lead_time(candidates, get_group_lead_time(group
    _id))

# 5) Scoring
scored = []
for s,e in candidates:
    score = (
        0.25*hour_pref(participants, s, tz_ref) +
        0.15*lead_time_component(s) +
        0.25*capacity_slack(venue_id, s, e) +
        0.15*venue_quality(venue_id) +
        0.20*group_harmony(group_id)
    )
    scored.append((s,e,score))

# 6) Retornar Top-N
return sorted(scored, key=lambda x: x[2], reverse=True)[:50]

```

Complexidade: varredura com **sweep line** e estruturas em listas ordenadas:
 $O(M \log M)$ $O(M \log M)$ $O(M \log M)$, onde M é o nº total de bordas de intervalos após expansão.

7) APIs (idempotentes, com controle de concorrência)

POST /api/schedule/propose

Body:

```
{
  "group_id": "g-123",
  "participants": ["u-1","u-2","u-3"],
  "venue_id": "v-99",      // opcional
  "duration_min": 90,
  "horizon_days": 14
}
```

Resp:

```
{
  "slots": [
    {"start_utc":"2025-08-22T18:00:00Z","end_utc":"2025-08-22T19:30:00Z","score":0.873},
    {"start_utc":"2025-08-23T15:00:00Z","end_utc":"2025-08-23T16:30:00Z","score":0.861}
  ]
}
```

POST /api/schedule/hold

Headers: Idempotency-Key: <uuid>

Body:

```
{
  "group_id": "g-123",
  "venue_id": "v-99",
  "start_utc":"2025-08-22T18:00:00Z",
  "end_utc":"2025-08-22T19:30:00Z",
  "expires_sec": 600
}
```

Resp: { "hold_id":"h-777","status":"hold","expires_at":"2025-08-22T17:50:00Z" }

POST /api/schedule/confirm

```
Body: { "hold_id":"h-777", "payment_token": null }  
Resp: { "booking_id":"b-555","status":"confirmed" }
```

```
DELETE /api/schedule/hold/h-777  
Resp: { "status":"released" }
```

Webhooks (parceiros):

- `POST /webhooks/partner/capacity_update` → atualiza `partner_capacity`.
- `POST /webhooks/partner/blackout_update` → insere/atualiza `partner_blackout`.

8) Concorrência e consistência

- **Locks distribuídos** por `(venue_id, window)` (Redis Redlock).
- Transações com `SELECT ... FOR UPDATE SKIP LOCKED` em `partner_capacity` para evitar sobre venda.
- **Hold TTL** (ex.: 10 min) com **worker** que expira e libera capacidade.
- **Idempotency-Key** obrigatório em `hold/confirm` para evitar duplicatas.

9) Casos-limite e validações

- **DST**: recusar slots em lacunas e desambiguar horários duplicados.
- **Noite virada**: permitir `[s,e)` cruzando meia-noite (ex.: 23:30–01:00).
- **Múltiplos viajantes**: interseção com **todas** as janelas de viagem.
- **Capacidade variável**: diferentes janelas com capacidades distintas no mesmo dia.
- **Cancelamentos**: liberar `holds/bookings` e recalcular top-N imediatamente.
- **Fusos heterogêneos**: sempre converter entradas locais para UTC **antes** de operar.

10) Segurança, privacidade e observabilidade

- **Least-privilege** em RBAC para leitura de calendários (apenas “busy/free”).
- **Criptografia**: TLS 1.3 em trânsito, AES-256 em repouso.
- **Logs e métricas**:

- Kafka: `schedule.events` → `PROPOSED|HOLD_PLACED|HOLD_EXPIRED|CONFIRMED|CANCELLED|CAPACITY_UPDATE`.
 - Prometheus/Grafana: latências, taxa de sucesso, conflitos de lock, taxa de overbooking (ideal = 0).
 - **Anti-scraping:** rate limiting por usuário/grupo nas rotas de consulta.
-

11) Testes (qualidade)

- **Unitários:** interseção/união de intervalos, conversões TZ, DST.
 - **Property-based:** aleatorizar calendários e validar invariantes (nenhum slot fora de janelas/blackouts).
 - **Carga:** 10k grupos concorrendo por slots numa mesma cidade.
 - **Chaos:** falhas no webhook de capacidade; garantir consistência após reprocesso.
-

12) Exemplo prático (simplificado)

- Grupo `g-123` (duração 90 min), participantes `u-1` (viajante) e `u-2` (local).
 - Viajante `u-1` → Lisboa (`Europe/Lisbon`), janela viagem: `[2025-08-22 00:00Z, 2025-08-25 00:00Z)`.
 - `u-1` disponível localmente 18:00–22:00; `u-2` 17:00–20:00 (ambos TZ Lisboa).
 - Interseção local → 18:00–20:00. Fatiando com 90 min → candidato `[18:00,19:30]`.
 - Parceiro aberto 17:00–22:00; capacidade suficiente; lead time ok.
 - Score calculado → 0.873 → **Top-1**.
-

Fechamento

Esta camada entrega a **capacidade operacional** de sair do “querer encontrar” para o “**temos horário viável e confirmado**”, respeitando tempo, fuso, capacidade e governança.

Com ela, **Modo Viagem**, **Bora** e **Locais Parceiros** passam a falar a mesma língua temporal.

CAMADA 20 — SISTEMA DE NOTIFICAÇÕES INTELIGENTES (BORA, VIAGEM, PARCEIROS, ALERTAS)

Entrada

Esta camada orquestra **toda a comunicação proativa** do ecossistema: avisos do **Modo Viagem**, atualizações dos **grupos Bora**, recomendações de **Locais Parceiros** e, principalmente, **alertas de segurança**.

O objetivo é entregar **a mensagem certa, no momento certo, pelo canal certo**, com **anti-spam, priorização, respeito às preferências do usuário** e **SLA rigoroso** — tudo sem vaziar dados sensíveis.

Parte Técnica (99%)

1) Arquitetura lógica

Componentes

- `notifications-engine` (core de orquestração e rankeamento)
- `template-service` (modelos i18n, placeholders, redaction)
- `user-prefs-service` (opt-in/out, quiet hours, canais permitidos)
- `budget-service` (rate limit/leaky-bucket por usuário e por evento)
- `channel-adapters` (push, in-app, e-mail; SMS reservado a guardiões)
- `scheduler` (agendamentos T-N, políticas DND e reenvio)
- `dedupe-service` (chaves idempotentes e supressão)
- `audit-log` (trilha LGPD/GDPR)
- `metrics` (Prometheus/Grafana)

Filas / tópicos (Kafka)

- `collisions.detected` (IA: Viagem)
- `bora.group.created|updated|member_joined|message_posted`
- `schedule.hold.placed|expired|confirmed` (Camada 19)
- `partner.capacity.updated|blackout.updated`

- `security.alert.raised` (botão de alerta, Camada 13)

Armazenamento

- PostgreSQL (estado, templates, entregas): `notification_template`, `notification_event`, `notification_queue`, `delivery_attempt`, `user_prefs`, `quiet_hours`, `device_token`
- Redis (baixa latência): `dedupe:*`, `budget:*`, `locks:*`, `scheduled:*`

2) Tipos de notificação (catálogo)

Tipo	Origem	Exemplo	Sensibilidade	Canal padrão
Colisão Preditiva	<code>collisions.detected</code>	"2 conexões previstas em Lisboa (semana 20–27)."	Média (sem dados exatos)	Push + In-app
Atualização Bora	<code>bora.*</code>	"Novo membro entrou / local alterado."	Baixa/Média	In-app; Push se alta prioridade
Slot & Hold	<code>schedule.hold.*</code>	"Sua reserva expira em 10 min."	Alta	Push (prioritário)
Parceiro Seguro	<code>partner.*</code>	"Local parceiro próximo disponível."	Baixa	In-app recomendação
Alerta de Segurança	<code>security.alert.raised</code>	"Ajuda em andamento; guardião notificado."	Crítica	Push + SMS guardião
Resumo Diário	batch	"Sugestões e convites do dia."	Baixa	In-app digest / e-mail (opt-in)

Regra de privacidade: notificações **não exibem coordenadas, hotel, datas exatas ou nomes de terceiros**; abrem deep-link para tela segura no app.

3) Preferências, DND e orquestração de canais

Preferências do usuário (`user_prefs`)

- `channels_allowed` : `{ push: true, inapp: true, email: false, sms: false }`
- `event_types` : granular (ex.: colisões on/off)

- `quiet_hours` : janela local por TZ (ex.: 22:00–08:00)
- `sensitive_mode` : “aproximar datas/locais” (default ON)

Políticas

- **DND**: durante quiet hours, **suprimir push**, reprogramar para próxima janela; exceções: `security.alert` , `hold.expiring < 10min` .
- **Fallback de canal**: Push falhou → In-app; se evento crítico e usuário habilitou email → e-mail.
- **Opt-out por tipo**: respeitar por evento (ex.: sem push para parceiros).

4) Anti-spam e orçamento de entrega (rate limiting)

Modelo leaky-bucket por usuário

- `daily_budget` (ex.: 8 tokens/dia)
- `burst_budget` (ex.: 3 por 30 min)
- Custo por severidade: crítica=3, alta=2, média=1, baixa=0.5
- Redis keys: `budget:{user_id}:day` , `budget:{user_id}:window`

Deduplicação

- Chave idempotente: `hash(user_id|type|context|period)`
- TTL dedupe: 24h (colisões); 2h (Bora updates); 15m (hold reminder)
- Política “coalesce”: várias mudanças próximas → **1 notificação consolidada**

5) Priorização e rankeamento

Score final (0–1):

$$S = 0.35 \cdot \text{Rel} + 0.25 \cdot \text{Urg} + 0.15 \cdot \text{Relac} + 0.15 \cdot \text{Harm} + 0.10 \cdot \text{BV} - \text{Pen}$$

$$S = 0.35 \cdot \text{Rel} + 0.25 \cdot \text{Urg} + 0.15 \cdot \text{Relac} + 0.15 \cdot \text{Harm} + 0.10 \cdot \text{BV} - \text{Pen}$$

$$S = 0.35 \cdot \text{Rel} + 0.25 \cdot \text{Urg} + 0.15 \cdot \text{Relac} + 0.15 \cdot \text{Harm} + 0.10 \cdot \text{BV} - \text{Pen}$$

- `Rel` (Relevância): normalização dos scores das Camadas 12/15 (colisão/harmonia/afinidade).
- `Urg` (Urgência): $\text{Urg} = \min(1, \frac{1}{1 + \ln(1 + \Delta t_{\text{hrs}})})$ $\text{Urg} = \min(1, \frac{1}{1 + \ln(1 + \Delta t_{\text{hrs}})})$; quanto menor o tempo, maior o valor.

- **Relac** (Relação): 1.0 se conexão autêntica / mesmo Bora; 0.5 se Premium verificado; 0.0 caso contrário.
- **Harm** (Harmonia de grupo): de 0–1 (Camadas 12/15).
- **BV** (Business Value): 0–1 (ex.: parceiro verificado e “zona segura” próxima → até 0.3).
- **Pen** (Penalidades):
 - Quiet hours (se não crítico): +0.3
 - Saturação de mensagens recentes: +0.2
 - Repetição (dedupe “soft”): +0.1

Buckets de prioridade

- $S \geq 0.8$ → **Alta** (push imediato, ignora DND se crítico)
- $0.6 \leq S < 0.8$ → **Média** (push respeitando DND, + in-app)
- $0.4 \leq S < 0.6$ → **Baixa** (in-app / digest)
- < 0.4 → suprimir

6) Pipeline de processamento

1. **Ingest** (Kafka) → enriquecimento (perfil, TZ, preferências).
2. **Dedup** (Redis) → aborta se chave existe.
3. **Budget check** (Redis leaky-bucket) → aborta ou reprograma.
4. **Scoring & Routing** → bucket + canal.
5. **Template render** (i18n + placeholders; sem dados sensíveis).
6. **Entrega** (adapter) com **backoff exponencial** (max 3 tentativas).
7. **Ack/metrics/audit** → Prometheus + PostgreSQL.

Outbox pattern: para eventos próprios (ex.: schedule), persistir em `notification_outbox` e publicar transacionalmente.

7) Templates (exemplos, i18n)

Colisão (Viagem)

- *Título:* “Conexões previstas no seu destino”

- *Body*: "Há 2 pessoas com alta compatibilidade na **região** e no **período** indicado. Veja no app."
- **Sem datas exatas**; deep-link: `app://travel/collisions?trip=...`

Hold (agendamento)

- *Título*: "Sua reserva expira em 10 min"
- *Body*: "Confirme o horário proposto ou ele será liberado."
- Deeplink: `app://schedule/review?hold=...`

Bora (update)

- *Título*: "Local atualizado pelo host"
- *Body*: "O ponto de encontro foi ajustado. Veja os detalhes."

Alerta de segurança

- *Título*: "Alerta recebido — suporte ativado"
- *Body*: "Estamos atuando. Você pode compartilhar sua rota com um guardião."
- Guardião (SMS): link de rastreo temporário (token curto, expira em 60 min)

8) APIs (idempotentes)

```
POST /api/notifications/send
Headers: Idempotency-Key: <uuid>
Body:
{
  "user_id": "u123",
  "type": "travel.collision",
  "context": { "trip_id": "t789", "approx_period": "meados de agosto", "region": "Lisboa" },
  "severity": "high",
  "preferred_channel": "push"
}
```

```
POST /api/notifications/schedule
{
  "user_id": "u123",
  "type": "schedule.hold.expiring",
  "run_at": "2025-08-19T19:50:00Z",
}
```

```

"context": { "hold_id": "h777", "minutes_left": 10 }
}

GET /api/notifications/inbox?cursor=... // in-app feed
GET /api/notifications/preferences
PUT /api/notifications/preferences
{
  "channels": { "push": true, "inapp": true, "email": false, "sms": false },
  "quiet_hours": { "start": "22:00", "end": "08:00", "tz": "America/Sao_Paulo"
},
  "event_types": { "collisions": true, "bora_updates": true, "partners": false }
}

POST /api/notifications/ack
{ "delivery_id": "d555", "status": "seen" }

```

Esquema `delivery_attempt`

```

CREATE TABLE delivery_attempt (
  delivery_id UUID PRIMARY KEY,
  user_id UUID NOT NULL,
  type TEXT NOT NULL,
  channel TEXT NOT NULL,      -- push|inapp|email|sms
  status TEXT NOT NULL,      -- sent|failed|seen|suppressed|scheduled
  severity TEXT NOT NULL,
  score NUMERIC(4,3) NOT NULL,
  created_at TIMESTAMPTZ DEFAULT NOW(),
  updated_at TIMESTAMPTZ DEFAULT NOW(),
  meta JSONB
);
CREATE INDEX ON delivery_attempt (user_id, created_at DESC);

```

9) Agendamentos e reenvios

- **Hold expiring:** agendar T-10 e T-2 min (se não confirmado).
- **Pré-viagem:** 48h antes do início da janela (resumo), e 4h antes (checklist).

- **Bora:** lembrete T-60 / T-15 (apenas se presença confirmada).
- **Reenvio:** backoff exponencial 5s/30s/2min; mover para DLQ após falha.

Redis scheduler: `zset scheduled:{shard}` com score = epoch; workers fazem `ZRANGEBYSCORE` e disparam.

10) Segurança e privacidade

- **Redaction:** placeholders sensíveis (datas/locais) substituídos por versões aproximadas quando `sensitive_mode=true`.
 - **Sem PII de terceiros** em push; identidades só dentro do app após consentimento mútuo.
 - **Criptografia:** TLS 1.3 em trânsito; AES-256 em repouso; KMS com rotação.
 - **LGPD/GDPR:** `audit_log` por evento com base legal e finalidade; retenção mínima necessária (ex.: 12 meses).
-

11) Observabilidade e SLO

SLOs

- 95% dos **push** prioridade alta em **< 2s**
- 99% dos **holds expiring** entregues **≥ 8 min** antes do vencimento
- 0% de **over-notify** crítico durante quiet hours

Métricas

- taxa de envio por tipo/canal, latência p50/p95, falhas por causa, budget drops, dedupe hit-rate, CTR por bucket (A/B).

Alertas operacionais

- queda >20% CTR em colisões (24h)
 - backlog `scheduled:*` > limiar
 - falha de adapter por canal > 2% (5 min)
-

12) Testes e qualidade

- **Unitários:** scoring, dedupe, budget, quiet hours, redaction.
- **Integração:** ingest Kafka → engine → adapter → ack.

- **Carga:** picos simultâneos (ex.: 50k “holds expiring” em horário cheio).
- **Chaos:** indisponibilidade de adapter; garantir DLQ e reprocesso.
- **A/B:** pesos (Rel/Urg/Relac/Harm/BV), janelas de DND, coalescing.

13) Pseudocódigo (rankeamento + entrega)

```
def process_event(evt):
    if dedupe.exists(evt.key):
        return "suppressed"

    prefs = load_user_prefs(evt.user_id)
    if not prefs.allows(evt.type):
        return "suppressed"

    if in_quiet_hours(now(), prefs) and not evt.is_critical():
        schedule_after_quiet(evt); return "scheduled"

    if not budget.consume(evt.user_id, cost=evt.severity_cost()):
        schedule_next_window(evt); return "budget_delayed"

    S = score(evt) # fórmula S acima
    channel = route(S, prefs, evt)

    payload = render_template(evt, redaction=prefs.sensitive_mode)
    result = deliver(channel, evt.user_id, payload)

    audit(evt, S, channel, result)
    return result
```

14) Casos-limite

- **Mudança de contexto** (ex.: hold confirmado antes do lembrete) → cancelar jobs pendentes.
- **Usuário sem device token** → cair para In-app; marcar push como “unreachable”.

- **Geodados ausentes** → não enviar recomendações de parceiros.
- **Saturação** → reduzir severidade para digest.
- **Eventos repetidos** → coalesce (ex.: 10 entradas no Bora em 5 min → 1 notificação "+9").

Fechamento

O **Sistema de Notificações Inteligentes** garante **relevância, precisão temporal e respeito ao usuário**, integrando Modo Viagem, Bora, Parceiros e Segurança em um pipeline **observável, auditável e anti-spam**. Ele transforma sinais do ecossistema em **ações concretas** que aumentam conversão, presença e segurança — sem comprometer privacidade.

CAMADA 22 — Backup, Recuperação de Desastres (DR) e Continuidade de Negócio

(Resiliência Total do Sistema de Conexões Reais — Viagem + Bora + Parceiros)

Entrada

Para um sistema que lida com **interações sociais reais, dados sensíveis e monetização direta**, a perda de dados ou a indisponibilidade do serviço pode ser fatal. Investidores exigem que FriendApp demonstre **resiliência corporativa**, com políticas sólidas de backup, recuperação de desastres (DR) e continuidade de negócio.

O objetivo desta camada é garantir que **mesmo diante de falhas graves (apagão de datacenter, corrupção de banco, ataque cibernético ou desastre natural)**, o ecossistema continue funcionando com **mínima interrupção e nenhuma perda crítica de dados**.

Estrutura Técnica

1. Política de Backup

- **Bancos Relacionais (PostgreSQL):**
 - *Full backup* diário.

- *Incremental backup* a cada 15 minutos via **WAL (Write Ahead Log)**.
 - **Firestore (NoSQL):**
 - Snapshots automáticos a cada hora.
 - Retenção mínima: 30 dias.
 - **Neo4j (Grafos):**
 - *Checkpoint* a cada 30 minutos + exportação em formato *GraphML*.
 - **Armazenamento Multi-Cloud:**
 - Backups replicados em **AWS S3 (Glacier Deep Archive)** e **GCP Cloud Storage Nearline**.
 - Dados criptografados em AES-256 + chaves gerenciadas por **KMS**.
-

2. Recuperação de Desastres (DR)

- **RPO (Recovery Point Objective):**
 - Máx. perda de dados: **5 minutos** (graças ao WAL streaming e Kafka para eventos).
 - **RTO (Recovery Time Objective):**
 - Tempo máximo de indisponibilidade: **30 minutos**.
 - **Zonas Ativas-Ativas:**
 - Kubernetes clusters rodando em **duas regiões distintas (ex: São Paulo + Virgínia)**.
 - Balanceamento via **Cloudflare Load Balancer**.
 - **Failover Automático:**
 - Detectado via **Prometheus + AlertManager**.
 - Redirecionamento automático para região saudável em < 60s.
-

3. Continuidade de Negócio

- **Ambiente Secundário "Hot-Standby":**
 - Réplica em tempo real dos serviços críticos:
 - `autenticacao-service` .
 - `verificacao-service (DUC/DCO)` .

- `conexoes-reais-service` .
- `economia-service` .

- **Serviços não críticos (suporte, comunidade, gamificação):**
 - Podem operar em modo *degraded* durante DR, mas nunca paralisar completamente.
- **Testes de Continuidade:**
 - Exercícios de *Chaos Engineering* com **Gremlin** ou **LitmusChaos**.
 - Simulação trimestral de desastre completo (ex: perda de um datacenter).

4. Fluxo Operacional

1. Detecção automática da falha (Prometheus alerta → PagerDuty).
2. Failover para cluster secundário (Cloudflare → Kubernetes).
3. Replicação de logs em tempo real para manter integridade de auditoria.
4. Retorno ao cluster original após recuperação, com sincronização automática de deltas.

Modelo Matemático de SLA de Resiliência

Disponibilidade Global (%):

$$A_{global} = 1 - (MTTR / (MTTR + MTBF))$$

$$A_{global} = 1 - (MTTR / (MTTR + MTBF))$$

- **MTTR** = Mean Time To Recovery (meta = 30 min).
- **MTBF** = Mean Time Between Failures (meta = 180 dias).

 Com esses valores:

$$A_{global} = 1 - (0.5 / (0.5 + 4320)) \approx 99.99\%$$

$$A_{global} = 1 - (0.5 / (0.5 + 4320)) \approx 99.99\%$$

Ou seja, **4 minutos de downtime permitido por mês.**

Fechamento

Esta camada garante que o FriendApp seja visto não apenas como um app social, mas como uma **infraestrutura corporativa com nível bancário de resiliência**.

Mesmo diante de falhas críticas, o sistema:

- ✅ Se recupera em até 30 minutos.
- ✅ Perde no máximo 5 minutos de dados.
- ✅ Mantém 99.99% de disponibilidade global.

CAMADA 23 — Pipeline de Entrega Contínua (CI/CD), Automação de Testes e Estratégias de Rollback

(Garantia de Evolução Segura e Ininterrupta do Sistema de Conexões Reais — Viagem + Bora + Parceiros)

Entrada

O FriendApp é uma plataforma viva, que precisa receber **novas funcionalidades semanais** sem comprometer estabilidade. Para investidores, a pergunta é clara: *“Como vocês garantem que novas features não quebrem o sistema em produção?”*.

Esta camada estabelece o **pipeline de integração e entrega contínua (CI/CD)**, combinando testes automatizados, monitoramento ativo e estratégias seguras de rollback.

Estrutura Técnica

1. Pipeline CI/CD — Fluxo Geral

- **Controle de Versão:**
 - Git + Git Flow (branches: `main`, `develop`, `feature/*`, `hotfix/*`).
- **CI (Continuous Integration):**
 - GitHub Actions ou GitLab CI para build, testes unitários e análise estática.
 - Cada `commit` em `feature/*` dispara validações automáticas.

- **CD (Continuous Delivery/Deployment):**
 - ArgoCD + Helm Charts para deploy em clusters Kubernetes.
 - Releases em *waves* (blue/green + canary).
-

2. Testes Automatizados

- **Unitários:** Cobertura mínima 80%.
 - **Integração:** Validação entre microserviços (`conexoes-reais-service` ↔ `economia-service`).
 - **Contratos de API:** Pact tests para garantir consistência entre front e back.
 - **E2E (End-to-End):** Cypress/Playwright validando jornadas críticas:
 - Criação de grupo Bora.
 - Ativação de Modo Viagem.
 - Integração com Local Parceiro.
 - **Testes Não Funcionais:**
 - Performance (k6/JMeter).
 - Segurança (OWASP ZAP, Snyk).
-

3. Deploy Progressivo

- **Blue/Green Deployment:**
 - Duas versões simultâneas (ativa e standby).
 - Troca instantânea em caso de falha.
 - **Canary Releases:**
 - 5% dos usuários recebem a nova versão.
 - Expansão gradual → 25% → 50% → 100% se métricas forem saudáveis.
 - **Feature Flags:**
 - Habilitar/desabilitar funcionalidades via **LaunchDarkly** sem precisar redeploy.
-

4. Monitoramento e Telemetria

- **Logs:** Stack ELK (Elasticsearch + Logstash + Kibana).

- **Métricas:** Prometheus + Grafana.
 - **Acompanhamento em Produção:**
 - Latência de APIs < 200ms (p95).
 - Erros < 0.1% por endpoint.
 - Tempo médio de build + deploy < 20 minutos.
-

5. Rollback

- **Automático:**
 - Se erro $\geq 2\%$ ou latência > 500ms por 5 min consecutivos, rollback automático.
 - **Manual:**
 - Botão de rollback no painel de DevOps.
 - Retorno para última versão estável em < 2 minutos.
-



Fórmula de Risco em Deploy

$R_{\text{deploy}} = P_{\text{falha}} \times \text{Impacto}_{\text{usuário}} \times R_{\text{deploy}} = P_{\text{falha}} \times \text{Impacto}_{\text{usuário}}$

$R_{\text{deploy}} = P_{\text{falha}} \times \text{Impacto}_{\text{usuário}}$

- P_{falha} = probabilidade baseada em cobertura de testes.
- $\text{Impacto}_{\text{usuário}}$ = nº de usuários atingidos pela nova versão.

➡ Releases **canary** reduzem $\text{Impacto}_{\text{usuário}}$, minimizando R_{deploy} .



END Fechamento

Com este pipeline:

- ✓ Releases semanais (ou diárias) com risco quase zero.
- ✓ Qualquer bug é revertido em até 2 min.
- ✓ Desenvolvedores focam em inovação, não em apagar incêndios.

CAMADA 24 — Estratégia de Testes Avançados

(Garantia de Qualidade Total para Viagem + Bora + Parceiros)

Entrada

O pipeline de entrega (Camada 23) garante releases rápidos, mas sem uma **estratégia de testes profunda**, a velocidade pode virar risco.

Esta camada define um **ecossistema completo de validação** — garantindo que cada interação, API, fluxo de usuário e integração externa seja **matematicamente testado, medido e aprovado** antes de chegar ao usuário final.

Tipos de Testes

1. Testes Unitários

- Cobertura mínima: **80% do código back-end e 70% do front-end**.
- Frameworks: Jest (Node.js), PyTest (Python), JUnit (Java).
- Exemplo: Validar função `calcular_score_harmonia()` retorna valor entre 0–100 em todos os cenários.

2. Testes de Integração

- Validar comunicações entre microsserviços.
- Exemplos críticos:
 - `conexoes-reais-service` ↔ `economia-service` (FriendCoins para ativar Viagem Premium).
 - `bora-service` ↔ `partners-service` (reserva de espaço em Local Parceiro).
- Ferramentas: Postman/Newman + Testcontainers.

3. Testes de Contrato (APIs)

- Garantir que mudanças em um serviço não quebrem outro.
- Pact tests: definem contratos para `/api/viagem/projetar`, `/api/bora/create`, `/api/partners/checkin`.

- Validação automática no CI a cada commit.
-

4. Testes End-to-End (E2E)

- Simulação real da jornada do usuário:
 - Criar viagem → projetar energia → receber notificações → entrar em grupo Bora → checar Local Parceiro.
 - Ferramentas: Cypress + Playwright.
 - Cenários críticos:
 - Usuário não-verificado tentando criar Bora (deve falhar).
 - Colisão energética detectada → notificação em < 2s.
-

5. Testes de Performance

- Ferramentas: k6, JMeter.
 - Métricas:
 - `latência_p95` < 200ms para APIs críticas.
 - Suportar **100k usuários simultâneos** em pico de viagens.
 - Kafka processando ≥ 10k eventos/seg.
-

6. Testes de Segurança

- OWASP Top 10 validado em cada release.
 - Ferramentas: OWASP ZAP, Burp Suite, Snyk.
 - Casos cobertos:
 - SQL Injection (ex: inputs de locais).
 - Cross-Site Scripting (XSS em mensagens do Bora).
 - Autorização: somente criador do grupo pode expulsar membros.
-

7. Testes de Resiliência (Chaos Engineering)

- Ferramenta: Gremlin.
- Exemplos:
 - Desligar nó de Kubernetes → sistema deve reequilibrar.

- Derrubar `verificacao-service` → criação de Bora deve bloquear corretamente.

Fórmula de Confiabilidade

Confiabilidade_{release} = 1 -

(F_{unit} + F_{integr} + F_{E2E} + F_{perf} + F_{seg})
Confiabilidade_{release} = 1 - (F_{unit} + F_{integr} + F_{E2E} + F_{perf} + F_{seg})

Confiabilidade_{release} = 1 - (F_{unit} + F_{integr} + F_{E2E} + F_{perf} + F_{seg})

- Onde cada `F` = taxa de falha residual após a bateria de testes.
- Meta: **Confiabilidade ≥ 99.5%** antes de liberar produção.

Fechamento

Com essa arquitetura de testes:

- ✓ O sistema é seguro, rápido e previsível.
- ✓ Bugs são detectados antes de chegar ao usuário.
- ✓ A confiança dos investidores é sustentada por métricas objetivas de qualidade.

CAMADA 25 — Modelo de Dados Ampliado (ERDs e Estrutura Híbrida)

Entrada

Até aqui definimos APIs, lógica de matching, notificações e testes. Agora é hora de consolidar o **núcleo estrutural dos dados**:

- Entidades principais.
- Relacionamentos (1:N, N:N).
- Estratégia híbrida (SQL + NoSQL + Grafos).
- Índices e constraints para performance e integridade.

Entidades Principais

1. Usuário

Campos principais (PostgreSQL):

- `id_usuario` (UUID, PK)
- `nome` , `email` , `senha_hash` , `data_nascimento`
- `status_verificacao` (enum: pendente/aprovado/recusado)
- `nivel_energia` (float 0-100)
- `plano` (free/premium)

Relacionamentos:

- 1:N com Viagens.
 - 1:N com Grupos Bora (pode ser membro de muitos, criador de alguns).
 - N:N com Locais Parceiros (via check-ins).
-

2. Viagem

Campos (PostgreSQL + Firestore para status em tempo real):

- `id_viagem` (UUID, PK)
- `id_usuario` (FK)
- `destino` (cidade, país)
- `data_inicio` , `data_fim`
- `status` (ativa/futura/concluída)
- `visibilidade` (privada/conexões/premium/pública)

Relacionamentos:

- 1:N com notificações de colisão energética.
 - N:N com Parceiros (check-ins em eventos locais).
-

3. Grupo Bora

Campos (Firestore para chat + PostgreSQL para metadata):

- `id_bora` (UUID, PK)
- `id_criador` (FK)
- `titulo` , `descricao`

- `local_sugerido`
- `status (ativo/encerrado)`
- `score_harmonia (float)`

Relacionamentos:

- N:N com Usuários (tabela intermediária `participacao_bora`).
- 1:N com Locais Parceiros (opcionais).
- Logs de chat → Firestore.

4. Parceiro (Local)

Campos (PostgreSQL):

- `id_parceiro (UUID, PK)`
- `nome`, `endereco`, `geo_lat`, `geo_long`
- `categoria (bar, café, coworking, centro_holistico)`
- `plano (essencial/premium/visionário)`

Relacionamentos:

- N:N com Viagens (usuários fazendo check-in).
- N:N com Boras (grupos realizados no local).

Relacionamentos Chave

Usuário ↔ Viagem

- 1:N.
- Índice composto (`id_usuario`, `data_inicio`) para consultas rápidas.

Usuário ↔ Bora

- N:N via `participacao_bora (id_usuario, id_bora, role: criador/membro)`.
- Constraint: `UNIQUE(id_usuario, id_bora)` evita duplicidade.

Bora ↔ Parceiro

- N:N via `bora_parceiro (id_bora, id_parceiro)`.
- Usado para mapear encontros em locais parceiros.

Usuário ↔ Parceiro

- N:N via `checkin (id_usuario, id_parceiro, timestamp)`.
- Armazenado também em Firestore para leitura em tempo real.



Modelo de Grafos (Neo4j)

Exemplo de nós e arestas:

- Nós: `Usuário`, `Viagem`, `Bora`, `Parceiro`.
- Arestas:
 - `(Usuário)-[:PARTICIPA]→(Bora)`
 - `(Usuário)-[:CHECKIN]→(Parceiro)`
 - `(Viagem)-[:VINCULA]→(Parceiro)`
 - `(Usuário)-[:CONEXAO_AUTENTICA]→(Usuário)`

Consulta de exemplo (Cypher):

```
MATCH (u:Usuario)-[:PARTICIPA]→(b:Bora)←[:PARTICIPA]-(outro:Usuario)
WHERE u.id = "123"
RETURN outro.nome, b.titulo, b.score_harmonia
```

👉 Retorna usuários que compartilham o mesmo Bora + score vibracional.



Diagrama ERD (simplificado em texto)

```
Usuário (1)——(N) Viagem
Usuário (N)——(N) Bora
Usuário (N)——(N) Parceiro
Bora (N)——(N) Parceiro
```



Índices e Performance

- `idx_viagem_destino`: consultas rápidas por destino.
- `idx_bora_status`: monitoramento de grupos ativos.

- `idx_checkin_timestamp` : análise em tempo real de fluxo de usuários em locais.
- Particionamento de Viagens por `data_inicio` → melhora queries históricas.

← **Fechamento**

Com esse modelo híbrido (SQL + NoSQL + Grafos):

- ✓ Temos consistência para dados críticos (PostgreSQL).
- ✓ Flexibilidade e escalabilidade para interações em tempo real (Firestore).
- ✓ Inteligência relacional profunda para matching e harmonia (Neo4j).

CAMADA 26 — Governança de Dados, Compliance e Políticas de Retenção

Entrada

Chegamos ao ponto em que o **Sistema de Conexões Reais (Modo Viagem + Bora + Parceiros)** precisa demonstrar não apenas inovação, mas também **responsabilidade jurídica e regulatória**.

Isso envolve:

- Proteção dos dados dos usuários.
- Conformidade com leis como **LGPD (Brasil)** e **GDPR (Europa)**.
- Definição de políticas claras de **retenção, backup e descarte**.
- Governança de dados integrada ao ciclo de vida do produto.

Princípios de Governança

1. Minimização de Dados

- Capturar apenas o essencial para a funcionalidade (ex.: dados de viagem = destino + período, mas nunca detalhes excessivos como número do voo).
- Evitar coleta de dados sensíveis sem base legal clara.

2. Consentimento Granular

- O usuário escolhe, em nível funcional, quais dados compartilha:
 - Viagens → só IA, conexões ou público.
 - Check-ins em Parceiros → só visíveis para amigos ou abertos ao feed.
- Consentimento armazenado com **hash temporal** (prova de aceite).

3. Transparência e Portabilidade

- Dashboard com:
 - Histórico de viagens.
 - Histórico de Boras participados.
 - Locais parceiros visitados.
- Exportação em JSON/CSV (API `/api/users/export`).
- Exclusão garantida via `/api/users/delete` com SLA de 48h.

Política de Retenção de Dados

- **Viagens concluídas** → Retenção por 24 meses (para estatísticas e recomendações futuras). Após isso, apenas dados anonimizados.
- **Grupos Bora encerrados** → Logs de chat e interações guardados por 12 meses (moderação). Após isso, descarte automático.
- **Check-ins em Locais Parceiros** → Retenção por 18 meses para fins de analytics.
- **Dados sensíveis (documentos DUC/DCO)** → Guardados criptografados por 5 anos, conforme normas de compliance KYC.

Segurança e Compliance

1. Criptografia

- **Em trânsito:** TLS 1.3 em todas as APIs.
- **Em repouso:** AES-256 para bancos PostgreSQL, Firestore e backups.
- **Segregação lógica** entre dados pessoais (PostgreSQL) e dados de interação social (Firestore/Neo4j).

2. Acesso Controlado (RBAC)

- Papéis: Admin, Moderador, Parceiro, Usuário.
- Restrições:
 - Moderador acessa apenas logs de denúncia.
 - Parceiros acessam apenas dados agregados (nunca PII).

3. Auditoria e Logs

- Logs imutáveis (blockchain-based ou append-only) para denúncias, alterações de consentimento e acessos administrativos.
- Retenção mínima de logs de segurança: 5 anos.



Backup e Continuidade

- **Backups diários:** PostgreSQL + Firestore.
- **RPO:** 15 minutos (perda máxima aceitável).
- **RTO:** 2h (tempo de recuperação).
- **Estratégia multi-cloud** (AWS + GCP) → failover automático via Kubernetes Federation.



Alinhamento com LGPD/GDPR

- **Base legal:** Consentimento (art. 7º, LGPD) + Legítimo interesse (para analytics).
- **Encarregado (DPO):** Definido no organograma.
- **Relatório de Impacto à Proteção de Dados (RIPD):** obrigatório antes de expansão internacional.
- **Direito ao Esquecimento:** Exclusão definitiva, exceto quando houver obrigação legal de retenção (ex.: documentos de verificação).



Fechamento

Com essa camada de **governança e compliance**:

- ✓ Garantimos confiança junto a usuários e investidores.
- ✓ Prevenimos riscos jurídicos (multas LGPD/GDPR).
- ✓ Criamos base sólida para internacionalização.

CAMADA 27 — CI/CD, VERSIONAMENTO E ESTRATÉGIAS DE ROLLBACK

(Pipeline de Entrega, Deploy Contínuo e Garantia de Estabilidade)

Entrada

A **Camada 27** trata da espinha dorsal do ciclo de vida de desenvolvimento: como o **FriendApp** é atualizado sem comprometer a experiência do usuário.

Aqui documentamos **CI/CD (Continuous Integration / Continuous Delivery)**, versionamento de código, estratégias de rollback e práticas para garantir que **nenhum deploy em produção quebre a harmonia vibracional do ecossistema**.

Componentes Principais

1. Integração Contínua (CI)

- **Ferramentas sugeridas:** GitHub Actions, GitLab CI ou CircleCI.
- **Pipeline padrão:**
 1. Commit no branch → dispara build automatizado.
 2. Execução de testes unitários, de integração e segurança.
 3. Validação automática do estilo de código (linters).
 4. Geração de artefatos versionados (containers Docker).

2. Entrega Contínua (CD)

- Deploys frequentes e automatizados em ambientes de **staging** e **produção**.
- **Estratégia recomendada:**
 - **Staging Mirror:** staging idêntico a produção, simulando tráfego real.
 - **Blue/Green Deployment:** duas versões rodando lado a lado; usuários são migrados gradualmente.
 - **Canary Releases:** liberar para 5% → 20% → 100% da base, garantindo rollback rápido se houver falhas.

3. Versionamento de Código

- **Estratégia: Git Flow adaptado.**
 - `main` : branch de produção.
 - `develop` : branch de integração.
 - `feature/*` : novas funcionalidades.
 - `hotfix/*` : correções emergenciais.
- **Versionamento Semântico (SemVer):**
 - `MAJOR.MINOR.PATCH` (ex.: 2.3.5).
 - **MAJOR**: mudanças que quebram compatibilidade.
 - **MINOR**: novas funcionalidades compatíveis.
 - **PATCH**: correções e ajustes menores.

4. Estratégias de Rollback

- **Rollback instantâneo:**
 - Se >3% dos usuários apresentarem falhas críticas no canário, rollback automático.
- **Snapshots de banco de dados:**
 - Antes de qualquer migração, snapshot é feito em PostgreSQL, Firestore e Neo4j.
- **Rollback de infraestrutura:**
 - Kubernetes → `kubectl rollout undo`.
 - Automatizado no pipeline para ambientes críticos.

5. Observabilidade Integrada ao Deploy

- **Monitoramento pós-deploy:**
 - Logs centralizados no ELK (Elasticsearch, Logstash, Kibana).
 - Métricas (Prometheus + Grafana).
 - Alertas automáticos no PagerDuty/Slack em caso de degradação.



Segurança no CI/CD

- **Secrets Management:** uso de Vault para chaves e tokens, nunca expostos no pipeline.
 - **Code Scanning:** SonarQube ou Snyk para checar vulnerabilidades antes do deploy.
 - **Política de Aprovação:** merges em `main` só são permitidos após 2 revisões + testes passarem.
-

← **Fechamento**

Com essa camada, o FriendApp estabelece uma **linha de produção confiável, ágil e resiliente**, capaz de suportar a evolução rápida da plataforma sem comprometer a estabilidade.

CAMADA 28 — ESTRUTURA DE TESTES AUTOMATIZADOS (UNITÁRIOS, CONTRATO, INTEGRAÇÃO, E2E, PERFORMANCE, SEGURANÇA, CAOS)

Entrada

Esta camada operacionaliza a estratégia de testes (definida nas Camadas 23–24) em **arquitetura, ferramentas, dados, ambientes e quality gates**. O objetivo é tornar o pipeline **determinístico e reproduzível**, reduzindo risco de regressão e garantindo SLOs antes de qualquer deploy.

Parte Técnica (99%)

1) Topologia de Ambientes e Isolamento

- **dev-local:** Docker Compose com serviços “fakes” (Kafka, PG, Redis, Firestore emulador).
- **ci-ephemeral:** ambiente efêmero por PR (namespace K8s isolado; bancos com sufixo do PR).
- **staging-mirror:** espelha produção (mesmas versões de serviços terceiros, feature flags espelhadas).

- **prod**: canário + blue/green.

Políticas:

- Dados de teste **sempre sintéticos** (sem PII).
- Sem tráfego real em staging (somente réplica de padrões de tráfego via replayers).

2) Pirâmide de Testes — Escopo e Ferramentas

Camada	Objetivo	Ferramentas	Gate
Unit	Funções puras, regras (score, datas)	Jest/PyTest/JUnit	≥80% cobertura backend, ≥70% frontend
Contrato	Estabilidade das APIs	Pact, OpenAPI Validator	0 falhas
Integração	Fluxos entre microserviços	Postman/Newman, Testcontainers	p95 < 200ms
E2E	Jornadas críticas	Playwright/Cypress	100% passos verdes
Performance	Latência/throughput	k6/JMeter	p95 < 200ms (core)
Segurança	OWASP/Deps	ZAP/Burp/Snyk	0 críticas
Caos	Resiliência	Gremlin/Litmus	SLOs mantidos

3) Gestão de Dados de Teste

- **Catalogo de Fixtures** (reusáveis) versionado:
 - `users_synthetic.json` (DUC/DCO válidos/falhos),
 - `travels_windows.json` (diferentes fusos, DST),
 - `bora_groups.json` (tamanhos/variação de harmonia),
 - `partners_capacity.json` (capacidade dinâmica/blackouts).
- **Data seeding** via migrations específicas de teste: `db/seed/test/*`.
- **Mascaramento/anonymização** automatizado para qualquer export importado para staging.

4) Testes Unitários — Pacotes Críticos

- **Matching**

- `cosineSimilarity()` , `groupHarmony()` , `collisionScore()` — testes property-based (Hypothesis/fast-check).
 - **Agenda (Camada 19)**
 - Interseções de intervalos, conversão TZ/DST, buffers, lead time.
 - **Notificações (Camada 20)**
 - Scoring S, budget (leaky-bucket), dedupe, DND.
 - **Segurança em tempo real (Camada 13)**
 - Botão de alerta: SLA < 5s (simulado com fakes de adapters).
-

5) Testes de Contrato (Consumer-Driven)

- **OpenAPI**: validação de esquemas e exemplos.
 - **Pact**: consumidores (app/web) definem contratos para:
 - `/api/travel/projection` , `/api/match/*` , `/api/schedule/*` , `/api/security/*` , `/api/partners/*` .
 - **CI Gate**: PR bloqueado se contrato quebrar; versionamento **SemVer** de esquemas.
-

6) Testes de Integração (com Testcontainers)

- Subir **PostgreSQL, Redis, Kafka, MinIO (S3)** em containers por suíte.
 - Casos:
 - `schedule.propose → hold → confirm` com lock distribuído.
 - Atualização de `partner_capacity` via webhook + recalculando slots.
-

7) E2E (Playwright/Cypress)

Cenários obrigatórios:

1. **Viagem privada (IA only)** → colisões calculadas, sem vazamento visual.
2. **Criar Bora (host verificado)** → convidar, aprovar, silenciar, encerrar.
3. **Zonas seguras** → sugestão de parceiro + reserva com hold e expiração.
4. **Alerta de segurança** → push para moderação + SMS guardião (mock).
5. **Privacidade** → projeção pública mostra apenas "região" e "janela temporal".

Infra: runner paralelo (sharding) + gravação de vídeo + captura de HAR.

8) Performance & Capacidade (k6)

- **Objetivos**
 - Core APIs (matching, schedule, notifications enqueue) p95 < 200ms, error rate < 0.1%.
 - Kafka: ≥10k msgs/s processadas; lag < 2s.
 - **Cenários**
 - "Pico de pré-viagem": 50k colisões em 10min.
 - "Janelão de expiração": 30k holds expiram em 5min.
 - **Infra auxiliar:** métricas Prometheus exportadas como gates no CI.
-

9) Segurança Automatizada

- **SAST:** SonarQube/Snyk (PR gate).
 - **DAST:** OWASP ZAP em staging (no-auth e com-auth).
 - **Dependency Audit:** fail the build em CVE alta/critica.
 - **AutZ:** testes para papéis (Host/Mod/Membro/Parceiro/Admin).
 - **Headers e CSRF:** verificação automática (helmet, SameSite, CSP).
-

10) Caos e Falhas Controladas

- **Gremlin/Litmus** in staging-mirror:
 - Queda de nó Kafka → reprocessar outbox sem perda.
 - Latência 300ms no Redis → orçamento/anti-spam funciona.
 - Falha do `verificacao-service` → bloqueio de criação de Bora (graceful).
 - **SLO guardrail:** se SLOs violados, teste reprova e PR bloqueia.
-

11) Quality Gates (CI/CD)

- **Unit:** cobertura mínima + mutation score ≥ 60% (stryker).
- **Contrato:** 0 incompatibilidades.
- **Integração:** suite verde + latências p95 < 200ms.
- **E2E:** 100% jornadas críticas.
- **Perf:** KPIs atingidos.

- **Segurança:** 0 críticas; altas mitigadas.
- **Caos:** manter SLOs sob falhas injectadas.

Falhou um gate → **no deploy**.

12) Observabilidade dos Testes

- **Test Analytics:** armazenar resultados em `qa_results` (PostgreSQL) + dashboards Grafana:
 - Tendência de flakiness por suíte.
 - Tempo médio por job.
 - Mapa de falhas por módulo (matching, schedule, notifications, security).

Esquema simplificado

```
CREATE TABLE qa_results (  
  run_id UUID, suite TEXT, test_name TEXT, status TEXT,  
  duration_ms INT, env TEXT, commit_sha TEXT, created_at TIMESTAMPTZ D  
  EFAULT NOW()  
);  
CREATE INDEX ON qa_results (suite, created_at DESC);
```

13) Utilitários de Teste (APIs internas)

```
POST /api/_test/seed  
{ "fixtures": ["users_basic","travels_dst","partners_capacity_peak"] }  
  
POST /api/_test/toggle-flag  
{ "flag": "feature.bora.guardian_sms", "enabled": true }  
  
POST /api/_test/time-travel  
{ "iso": "2025-08-22T18:00:00Z" } // congela relógio do cluster de teste
```

Protegidas por IP allowlist + auth interna; nunca habilitar em prod.

14) Exemplo de Pipeline (pseudo YAML)

```
jobs:
  unit:
    steps: [install, lint, test:unit, coverage, upload-artifacts]
  contract:
    steps: [generate-openapi, pact-verify]
  integration:
    services: [postgres, redis, kafka]
    steps: [migrate, seed, test:integration]
  e2e:
    needs: [deploy:staging]
    steps: [seed, run:playwright, collect:vrt]
  perf:
    steps: [run:k6, assert:kpis]
  security:
    steps: [snyk:ci, zap:spider, zap:baseline]
  gates:
    needs: [unit,contract,integration,e2e,perf,security]
    steps: [assert:quality-gates]
  deploy:
    needs: [gates]
    steps: [canary, monitor, promote-or-rollback]
```

Fechamento

Esta camada entrega a **engenharia de qualidade operacional**: ambientes isolados, dados sintéticos, gates objetivos e automação ponta a ponta. Resultado: **previsibilidade de releases, segurança constante e zero “surpresa” em produção**.

CAMADA 29 — API REFERENCE (OpenAPI/Swagger) — Viagem + Bora +

Agenda + Parceiros + Segurança + Notificações

Entrada

Esta camada entrega a **documentação técnica executável** (padrão OpenAPI/Swagger) para os módulos: **Modo Viagem, Bora, Agenda, Parceiros, Segurança em Tempo Real, Notificações e Privacidade**. Inclui **autenticação, erros, rate-limit, idempotência, paginação, versionamento e webhooks**. É base direta para gerar SDKs, testes de contrato (Pact) e portais de APIs.

1) Convenções, Segurança e Cabeçalhos

Base URL: `https://api.friendapp.com/v1`

Auth: OAuth2 (Authorization Code / PKCE) → **JWT** assinado (RS256) no header

`Authorization: Bearer <token>`

Scopes (exemplos): `travel:read`, `travel:write`, `bora:manage`, `security:alert`, `partners:read`, `notify:write`, `profile:read`, `privacy:write`

Headers padrão

- `Authorization: Bearer <jwt>`
- `Idempotency-Key: <uuid>` (obrigatório em `POST` sensíveis: hold/confirm/alert/send)
- `X-Request-ID: <uuid>` (corr. e auditoria)
- `X-Client-Version: <semver>`

Paginação (cursor)

- Query: `limit` (1–200, default 50), `cursor` (string opaca)
- Resposta: `next_cursor` (ou ausente)

Rate limiting

- Padrão usuário: **120 req/min**; críticos (alert, hold/confirm): **30 req/min**
- Resposta inclui `X-RateLimit-Limit`, `X-RateLimit-Remaining`, `X-RateLimit-Reset`

Erros: Problem+JSON

```
openapi: 3.0.3
info:
  title: FriendApp — Conexões Reais API
```

```
version: "1.0.0"
servers:
  - url: https://api.friendapp.com/v1
components:
  securitySchemes:
    OAuth2:
      type: oauth2
      flows:
        authorizationCode:
          authorizationUrl: https://auth.friendapp.com/oauth/authorize
          tokenUrl: https://auth.friendapp.com/oauth/token
          scopes:
            travel:read: Ler viagens
            travel:write: Criar/alterar viagens
            bora:manage: Gerir grupos Bora
            schedule:write: Propor/segurar/confirmar slots
            security:alert: Enviar alertas
            partners:read: Ler parceiros/zonas
            notify:write: Enviar/schedule notificações
            privacy:write: Configurar privacidade
      headers:
        IdempotencyKey:
          schema: { type: string, format: uuid }
          required: true
          description: Garante idempotência em POSTs sensíveis
  schemas:
    Problem:
      type: object
      required: [type, title, status]
      properties:
        type: { type: string, format: uri }
        title: { type: string }
        status: { type: integer }
        code: { type: string, description: "Código de erro FriendApp" }
        trace_id: { type: string }
        detail: { type: string }
```


2) Modo Viagem (Projeção, Listagem e Colisões)

Criar/atualizar projeção

```
POST /travel/projection
Authorization: Bearer ...
Content-Type: application/json
Idempotency-Key: <uuid>
```

```
{
  "destino": { "city": "Lisboa", "country": "PT" },
  "window": { "start": "2025-09-10T00:00:00Z", "end": "2025-09-18T00:00:00Z" },
  "visibility": "connections|premium|public|private",
  "abstraction": { "geo": "bairro|zona|cidade", "date": "janela|dia" },
  "intent_tags": ["cura","expansao"]
}
```

200

```
{ "travel_id": "t_9f1c", "status": "active" }
```

Listar projeções visíveis (respeita escopos/privacidade)

```
GET /travel/projections?city=Lisboa&limit=50&cursor=...
```

Consultar colisões preditivas (ranqueadas)

```
GET /travel/collisions?travel_id=t_9f1c&min_score=0.6
```

200

```
{
  "items": [
```

```
{ "user_id": "u_123", "score": 0.82, "period": "meados de setembro", "region": "Baixa-Chiado" }  
],  
"next_cursor": "eyJvZmZzZXQiOjUw...=="  
}
```

Excluir projeção

```
DELETE /travel/projection/{travel_id}
```

3) Bora (Grupos, Membros, Moderação)

Criar grupo Bora (requer DUC/DCO aprovado)

```
POST /bora  
Authorization: Bearer ...  
Idempotency-Key: <uuid>
```

```
{  
  "title": "Respiração e Café",  
  "description": "Encontro leve no fim da tarde",  
  "duration_min": 90,  
  "tz_ref": "Europe/Lisbon",  
  "visibility": "public|connections|premium",  
  "initial_policy": { "secure_zone_required": true }  
}
```

Obter grupo

```
GET /bora/{group_id}
```

Entrar/solicitar entrada

```
POST /bora/{group_id}/join
```

Silenciar membro (host/mod)

```
POST /bora/{group_id}/silence  
Content-Type: application/json
```

```
{ "target_user_id": "u_789", "hours": 24, "reason": "spam" }
```

Remover membro (host)

```
POST /bora/{group_id}/remove  
{ "target_user_id": "u_789", "reason": "conduta_inadequada" }
```

Encerrar grupo

```
POST /bora/{group_id}/close
```

Denunciar (usuário, mensagem ou grupo)

```
POST /bora/report  
{ "group_id": "g_123", "target_user_id": "u_789", "kind": "usuario|mensagem|gr  
upo", "reason": "assedio" }
```

Listar membros (resumo sem PII sensível)

```
GET /bora/{group_id}/members?limit=100&cursor=...
```

4) Agenda (Propor → Hold → Confirm)

Gerar slots candidatos (Camada 19)

POST /schedule/propose
Idempotency-Key: <uuid>

```
{
  "group_id": "g_123",
  "participants": ["u_1","u_2","u_3"],
  "duration_min": 90,
  "venue_id": "v_99"    // opcional
}
```

200

```
{
  "slots": [
    {"start_utc":"2025-09-12T18:00:00Z","end_utc":"2025-09-12T19:30:00Z","score":0.873}
  ]
}
```

Colocar HOLD (reserva temporária de capacidade)

POST /schedule/hold
Idempotency-Key: <uuid>

```
{
  "group_id":"g_123",
  "venue_id":"v_99",
  "start_utc":"2025-09-12T18:00:00Z",
  "end_utc":"2025-09-12T19:30:00Z",
  "expires_sec": 600
}
```

```
}
```

Confirmar booking

```
POST /schedule/confirm
Idempotency-Key: <uuid>
{ "hold_id":"h_777" }
```

Liberar hold

```
DELETE /schedule/hold/{hold_id}
```

5) Parceiros (Zonas Seguras, Capacidade, Webhooks)

Listar Zonas Seguras próximas

```
GET /partners/safe-zones?city=São Paulo&radius_km=5
```

Consultar capacidade para janela

```
GET /partners/{venue_id}/capacity?start=2025-09-12T18:00:00Z&end=2025-09-12T19:30:00Z
```

Webhooks (assinados) — atualização de capacidade

```
POST /webhooks/partner/capacity_update
X-FriendApp-Signature: t=1692468200,v1=hex(hmac_sha256(secret, body))
```

```
{
  "venue_id":"v_99",
  "window": { "start":"2025-09-12T18:00:00Z","end":"2025-09-12T22:00:00Z" },
}
```

```
"capacity": 40,  
"bookings": 18,  
"holds": 2  
}
```

Webhooks — blackout de operação

```
POST /webhooks/partner/blackout_update  
{ "venue_id": "v_99", "window": { "start": "2025-09-15T00:00:00Z", "end": "2025-09-15T06:00:00Z" }, "reason": "manutencao" }
```

6) Segurança em Tempo Real (Alerta, Guardiões)

Botão de Alerta

```
POST /security/alert  
Idempotency-Key: <uuid>
```

```
{  
  "user_id": "u_123",  
  "context": "bora|travel",  
  "context_id": "g_123",  
  "geo": { "lat": -23.5617, "lng": -46.6559 },  
  "share_with_guardian": true  
}
```

Compartilhar localização com guardião

```
POST /security/share-location  
{ "user_id": "u_123", "guardian_id": "gu_55", "session": "g_123", "expires_sec": 3600 }
```

7) Notificações (Envio, Agendamento, Preferências)

Enviar (programático)

POST /notifications/send
Idempotency-Key: <uuid>

```
{
  "user_id": "u_123",
  "type": "travel.collision",
  "severity": "high",
  "context": { "trip_id": "t_9f1c", "approx_period": "meados de setembro", "region": "Lisboa" },
  "preferred_channel": "push"
}
```

Agendar

POST /notifications/schedule
{ "user_id": "u_123", "type": "schedule.hold.expiring", "run_at": "2025-09-12T19:20:00Z", "context": { "hold_id": "h_777", "minutes_left": 10 } }

Preferências

GET /notifications/preferences
PUT /notifications/preferences

```
{
  "channels": { "push": true, "inapp": true, "email": false, "sms": false },
  "quiet_hours": { "start": "22:00", "end": "08:00", "tz": "America/Sao_Paulo" },
  "event_types": { "collisions": true, "bora_updates": true, "partners": false },
  "sensitive_mode": true
}
```

Inbox e ACK

```
GET /notifications/inbox?limit=50&cursor=...
POST /notifications/ack { "delivery_id":"d_555","status":"seen" }
```

8) Privacidade (Visibilidade e Abstração)

```
POST /privacy/set_visibility
{ "feature":"modo_viagem", "visibility":"connections", "abstraction": { "geo":"zona", "date":"janela" } }
```

```
GET /privacy/get_visibility?feature=modo_viagem
```

9) Esquemas de Erro (Problem+JSON) e Códigos

HTTP	code (FriendApp)	Descrição	Ação sugerida
400	<code>bad_request</code>	Payload inválido	Corrigir schema
401	<code>unauthorized</code>	Token ausente/inválido	Reautenticar
403	<code>forbidden_scope</code>	Escopo insuficiente (ex: privacy:write)	Solicitar escopo adequado
404	<code>not_found</code>	Recurso inexistente	Verificar IDs
409	<code>conflict</code>	Conflito (double hold / booking)	Usar Idempotency-Key; reprocessar
412	<code>precondition_failed</code>	DUC/DCO pendente para criar Bora	Completar verificação
422	<code>validation_error</code>	Horário fora de janela/blackout	Ajustar parâmetros
429	<code>rate_limited</code>	Limite excedido	Respeitar <code>Retry-After</code>
500	<code>internal_error</code>	Falha inesperada	Retry com Idempotency-Key
503	<code>service_unavailable</code>	Degradação/DR em curso	Repetir com backoff

Exemplo de erro


```
{
  "type":"https://api.friendapp.com/problems/conflict",
  "title":"Conflito de reserva",
  "status":409,
  "code":"conflict",
  "trace_id":"c9d1e8c1-..."
}
```

10) Idempotência e Concorrência

- **Idempotency-Key** obrigatória em **POST** que criam efeito (hold, confirm, alert, send).
- Chave válida por **24h**; respostas repetidas retornam **mesmo corpo/HTTP**.
- **Concorrência** agenda/capacidade: locks distribuídos por `(venue_id, window)` ; **409 conflict** se disputa.

11) Versionamento e Depreciação

- Prefixo de versão: `/v1`.
- Cabeçalho depreciação: `Sunset`, `Deprecation: true`, `Link: <doc>` quando endpoint for substituído.
- Garantia de **12 meses** de suporte após anúncio de sunset.

12) Webhooks — Segurança

- Header `X-FriendApp-Signature: t=<ts>,v1=<hmac>`.
- Algoritmo: `HMAC-SHA256(secret, t + "." + raw_body)` ; recusar `|now - t| > 5 min`.
- Reentrega com **exponential backoff** (até 3 tentativas); 2xx confirma.

13) Exemplos de cURL

1) Projeção de Viagem

```
curl -X POST https://api.friendapp.com/v1/travel/projection \
-H "Authorization: Bearer $TOKEN" -H "Idempotency-Key: $(uuidgen)" \
-d '{"destino":{"city":"Lisboa","country":"PT"},"window":{"start":"2025-09-10T00:00:00Z","end":"2025-09-18T00:00:00Z"},"visibility":"connections","abstraction":{"geo":"zona","date":"janela"}}'
```

2) Hold de Agenda

```
curl -X POST https://api.friendapp.com/v1/schedule/hold \
-H "Authorization: Bearer $TOKEN" -H "Idempotency-Key: $(uuidgen)" \
-d '{"group_id":"g_123","venue_id":"v_99","start_utc":"2025-09-12T18:00:00Z","end_utc":"2025-09-12T19:30:00Z","expires_sec":600}'
```

3) Alerta de Segurança

```
curl -X POST https://api.friendapp.com/v1/security/alert \
-H "Authorization: Bearer $TOKEN" -H "Idempotency-Key: $(uuidgen)" \
-d '{"user_id":"u_123","context":"bora","context_id":"g_123","geo":{"lat":-23.5617,"lng":-46.6559},"share_with_guardian":true}'
```

Fechamento

A **Camada 29** consolida uma referência API **executável** e **auditable**: autenticação forte, contratos claros, erros padronizados, rate-limit, idempotência, paginação e webhooks assinados. Com isso, front-ends, parceiros e QA conseguem **consumir e validar** o sistema ponta a ponta.

CAMADA 30 — SEGURANÇA APLICATIVA & ACESSO

(OAuth2/OIDC + JWT, RBAC/ABAC, MFA, Rotação de Chaves, Secrets, Hardening)

Entrada

Esta camada define — de ponta a ponta — **como autenticamos, autorizamos e protegemos** cada chamada do ecossistema **Modo Viagem + Bora + Parceiros**. O objetivo é entregar **segurança corporativa**: autenticação moderna (OAuth2/OIDC), tokens **JWT** com **assinatura assimétrica (RS256)**, **MFA** e autorização **RBAC + ABAC**, com **políticas auditáveis**, **rotação de chaves**, **gestão de segredos** e **hardening** em APIs, front e infra.

1) Arquitetura de Autenticação (OAuth2/OIDC)

Fluxos OIDC/OAuth2 suportados

- **Authorization Code + PKCE (apps mobile/web SPA)** — padrão para usuários finais.
- **Client Credentials (serviço↔serviço)** — microserviços internos e integrações B2B.
- **Device Code** — TV/IoT (opcional).

Emissores/Metadados

- `issuer` : `https://auth.friendapp.com`
- `/.well-known/openid-configuration` (descoberta OIDC)
- `/.well-known/jwks.json` (JWKS) — chaves públicas rotativas (KID).

Tokens

- **Access Token: JWT RS256**, `exp` 15 min, `aud` = serviço, `scope` granular (ex.: `bora:manage`).
- **Refresh Token: opaco** (UUID), guardado no **PostgreSQL/Redis** com **rotação obrigatória** (detecta reuso). TTL: 30 dias (renovável).
- **ID Token** (OIDC): claims não sensíveis (sem PII desnecessária).

Claims padrão (access token)

```
{
  "iss": "https://auth.friendapp.com",
  "sub": "u_123",
  "aud": "api.friendapp.com",
  "exp": 1692476400,
  "iat": 1692475500,
  "scope": "travel:write bora:manage",
}
```

```
"roles": ["user","verified","premium"],
"amr": ["pwd","mfa"],    // métodos de autenticação
"jti": "1b3b6f4c-..."  // id único p/ revogação
}
```

Rotas Auth principais

- `GET /oauth/authorize` (PKCE)
- `POST /oauth/token` (troca code/refresh; client credentials)
- `POST /oauth/revoke` (revogar refresh/sessão)
- `POST /oauth/introspect` (somente serviços internos)
- `GET /.well-known/openid-configuration`
- `GET /.well-known/jwks.json`

2) Gestão de Chaves (JWKS) & Rotação

Padrão: chave privada em **HSM/KMS** (AWS KMS/GCP KMS). Pública exposta em **JWKS** com `kid`.

- **Algoritmo:** `RS256`.
- **Rotação:** a cada **90 dias** (ou imediato em incidente).
- **Sobreposição:** manter chave antiga **30 dias** no JWKS (para tokens ainda válidos).
- **Revogação:** lista `revoked_jti` + `kid` inválido → rejeição imediata.

Exemplo **JWKS**:

```
{
  "keys": [{
    "kid": "2025-07-k1",
    "kty": "RSA",
    "alg": "RS256",
    "use": "sig",
    "n": "mIlBljANBgkqhkiG9w0B...", "e": "AQAB"
  }]
}
```

```
}
```

3) Autorização: RBAC + ABAC (OPA)

RBAC (papéis)

- `user` , `verified` , `premium`
- `host` (criador de Bora)
- `moderator` (Bora)
- `partner_operator` , `partner_admin`
- `support_agent` , `security_analyst` , `admin`

ABAC (atributos) — decisões por atributos de recurso/usuário:

- Ex.: "somente o `host` pode remover membro do `group_id` onde `host_id == sub`"
- Ex.: "parceiro só lê **capacidade agregada** do próprio `venue_id`"

Motor de política: OPA (Open Policy Agent) em sidecar ou no API Gateway.

Exemplo **Rego** (remoção de membro Bora):

```
package friendapp.bora

default allow = false

allow {
  input.endpoint == "POST /bora/{group_id}/remove"
  some gid
  gid == input.path_params.group_id
  input.user.roles[_] == "host"
  data.groups[gid].host_id == input.user.sub
}
```

4) MFA (Autenticação de múltiplos fatores)

Métodos

- **TOTP** (RFC 6238) — Authenticator Apps.

- **WebAuthn/Passkeys** — preferencial.
- **SMS** (fallback) — restrito; com rate limit forte.

Quando exigir (step-up auth)

- Acesso a **dados sensíveis** (alterar privacidade, DUC/DCO, denunciar/banir).
- Ações **críticas** (encerrar Bora, revogar sessão, acionar alerta de segurança).
- **Risco elevado** (SRA > 0.75 — ver Camada 21).

APIs

- `POST /auth/mfa/enroll` (TOTP/WebAuthn)
- `POST /auth/mfa/verify` (step-up)
- `GET /auth/mfa/methods` (listagem)

5) Sessões, Revogação, Reuso de Refresh

- **Sessões**: por dispositivo, com **device fingerprint** (hash de userAgent + chaves do SO) e **idle timeout** 14 dias.
- **Revogação**: `POST /auth/sessions/revoke` (por device) ou **global**.
- **Reuso de refresh** (anti-theft): se um refresh antigo é usado após rotação → **invalida toda a árvore** dessa sessão e força novo login/MFA.

Tabelas (PostgreSQL)

```
CREATE TABLE refresh_token (
  token_id UUID PRIMARY KEY,
  user_id UUID NOT NULL,
  session_id UUID NOT NULL,
  parent_token UUID,
  created_at TIMESTAMPTZ DEFAULT now(),
  expires_at TIMESTAMPTZ,
  used_at TIMESTAMPTZ,
  revoked_at TIMESTAMPTZ
);
```

```
CREATE TABLE session (
  session_id UUID PRIMARY KEY,
  user_id UUID NOT NULL,
```

```
device_fingerprint TEXT,  
last_ip INET, last_seen TIMESTAMPTZ,  
mfa_level TEXT, -- none|totp|webauthn  
created_at TIMESTAMPTZ DEFAULT now(),  
revoked_at TIMESTAMPTZ  
);  
  
CREATE TABLE revoked_jti (jti UUID PRIMARY KEY, revoked_at TIMESTAMPTZ  
Z DEFAULT now());
```

6) Autenticação Serviço↔Serviço (S2S)

- **mTLS** obrigatório entre microserviços (TLS 1.3).
- **Identidade de workload: SPIFFE/SPIRE** (`spiffe://friendapp.svc/<service>`).
- **Política de autorização** no gateway/OPA por `spiffelD` .
- **Rotação de certificados** automatizada (cert-manager).

7) Proteção de Segredos

- **Vault** para segredos dinâmicos (credenciais DB com TTL).
- **KMS** para chaves de criptografia de dados (AES-256-GCM).
- **Rotação de segredos**: 90 dias (ou imediato em incidente).
- **Sem segredos em variáveis de pipeline**; usar *workload identity*.

8) Hardening de APIs & Front

Transporte

- **TLS 1.3 obrigatório**; HSTS `max-age=31536000; includeSubDomains` .
- Ciphers modernos (AES-GCM/CHACHA20-POLY1305).

HTTP/Web

- **CSP** (bloqueio de inline JS), **X-Frame-Options: DENY**, **X-Content-Type-Options: nosniff**.
- **CORS**: whitelist por origem; **preflight cache** mínimo.

- **Cookies:** `Secure; HttpOnly; SameSite=Lax/Strict` .
- **CSRF:** tokens com *double submit* para rotas stateful web.
- **Input validation:** schema-first (OpenAPI) + sanitização (anti-SSRF/XSS).
- **WAF/CDN (Cloudflare):** DDoS, bot mgmt, rate-limit comportamental.
- **AuthZ por endpoint: escopos mínimos** (principle of least privilege).

Contas/senhas

- Hash: **Argon2id** (m=256MB, t=3, p=2) ou scrypt modern.
- Lockout progressivo: 5 falhas → backoff; captcha adaptativo.
- Recuperação de conta com **MFA + e-mail** (sem perguntas inseguras).

9) Middleware de Verificação (pseudocódigo)

```
def verify_access(request):
    # 1) Extrair e validar JWT
    token = parse_bearer(request.headers["Authorization"])
    claims = jwks_verify_rs256(token, cache_ttl=10m, leeway=60s)
    assert claims["aud"] == "api.friendapp.com"
    assert now() < claims["exp"]

    # 2) Checar revogação (jti) e escopos
    if is_revoked(claims["jti"]): deny(401)
    if not has_required_scopes(claims["scope"], request.endpoint): deny(403)

    # 3) Step-up MFA para endpoints sensíveis
    if endpoint_requires_mfa(request.endpoint) and "mfa" not in claims.get("amr", []):
        require_mfa_stepup()

    # 4) ABAC (OPA)
    input_ctx = build_input(request, claims)
    if not opa_allow("friendapp.bora", input_ctx):
        deny(403)

    # 5) Rate-limit & DDoS guard
    rate_guard.consume(claims["sub"], request.endpoint)
```



```
# 6) Prossegue  
return allow()
```

10) APIs de Segurança Complementares

```
# Lista de sessões ativas do usuário  
GET /auth/sessions  
# Revogar sessão específica  
POST /auth/sessions/revoke { "session_id": "sess_123" }  
  
# Enrolar/validar MFA  
POST /auth/mfa/enroll { "method": "totp|webauthn" }  
POST /auth/mfa/verify { "otp": "123456" }  
  
# Rotação de chaves (admin)  
POST /auth/keys/rotate { "alg": "RS256" }  
  
# Exigir step-up em tempo de execução (admin policy)  
POST /auth/policies/stepup  
{ "endpoint": "POST /bora/{id}/close", "required_mfa": "webauthn" }
```

11) Erros e Sinais de Segurança

HTTP	code	Motivo	Ação
401	invalid_token	Token ausente/expirado/fraude	Reauth
401	mfa_required	Endpoint sensível sem MFA	Step-up
403	insufficient_scope	Escopo insuficiente	Solicitar escopo
403	policy_denied	Regra OPA negou (ABAC)	Rever papéis
409	refresh_reuse	Reuso de refresh antigo detectado	Forçar logout
429	rate_limited	Excedeu limites	Backoff
451	legal_hold	Restrição legal/compliance	Suporte

12) Observabilidade & Auditoria de Acesso

- **ID de correlação** (`X-Request-ID`) propagado.
 - **Audit log** para: login/logout, step-up, falhas, revogações, trocas de chave, mudanças de política.
 - **Redação** automática (sem PII/token em logs).
 - Alertas SIEM (Camada 21) para anomalias (geo, horário, repetição de falhas).
-

13) Playbooks de Incidente (resumo)

- **Suspeita de vazamento de chave** → rodar `keys/rotate` (imediato), invalidar `kid` anterior, notificar SIEM, forçar revalidação de sessões críticas.
 - **Ataque de credenciais** → lockout adaptativo, CAPTCHA, travar tentativas por ASN/IP, exigir MFA obrigatório temporário.
 - **Refresh reuse em massa** → invalidação por árvore de sessão + notificação ao usuário.
-

Fechamento

Com esta camada, o FriendApp consolida um **perímetro de segurança moderno e mensurável**: autenticação robusta (PKCE, mTLS), **autorização contextual (RBAC + ABAC via OPA)**, **MFA com step-up, tokens e chaves rotativas**, segredos protegidos, **hardening** em múltiplas camadas e **playbooks de resposta**. O resultado é **risco operacional baixo, conformidade auditável e confiança para escalar** Viagem + Bora + Parceiros em produção.