Manual Técnico — IA Aurah Kosmos (v2 • 76 Camadas)

📦 CAMADA 01 — Identidade & Sessões (Auth/Device)

🎯 Objetivo da Camada

Garantir a **identidade única do usuário** dentro do ecossistema FriendApp, estabelecendo um vínculo seguro de sessão entre dispositivo, conta e contexto.

É o ponto inicial da jornada técnica: sem identidade clara, não existe confiança, reputação ou conexão autêntica.

A camada também assegura que **dados pessoais sensíveis (PII)** sejam sempre segregados e que o usuário tenha controle sobre sua privacidade desde o primeiro acesso.

- **Redução de dimensionalidade**: PCA ou UMAP aplicados para performance.

🔑 Entradas

- `device_fingerprint`: assinatura única do dispositivo (gerada via hardware + software).
- `oauth_token`: token seguro de autenticação inicial (via e-mail, social login ou phone OTP).
- `duc_dco_status`: status da verificação documental (DUC/DCO).

📤 Saídas

- `user_id` (UUIDv4): identificador único do usuário no ecossistema.
- `session_id`: identificador da sessão atual (curta duração).
- `auth_scope`: escopo de acesso concedido (ex.: feed, chat, eventos).

📊 Regras Técnicas

- **Sessões curtas**: expiram em 24h, com refresh automático quando autorizado.
- **2FA opcional**: ativável pelo usuário, recomendada em acessos sensíveis (financeiro/Premium).
- **Segregação de PII**: dados pessoais isolados em repositório dedicado e criptografado.
- **Rotatividade de chaves**: tokens renovados periodicamente para reduzir risco de vazamento.

🔧 Exemplo de Contrato de API

Requisição

`POST /auth/login`

```
{
  "device_fingerprint": "abc123xyz",
  "oauth_token": "ya29.a0AV..."
}
```

Resposta

```
{
  "user_id": "6d71b6c2-84df-4de2-bc1d-9a8ab22a87d3",
  "session_id": "c22b9fa4-44a0-45cc-8f7f-31899d4b1e67",
  "auth_scope": ["feed", "chat", "events"]
}
```

🧪 Métricas de Observabilidade

- `login_latency_ms`: tempo médio de resposta da autenticação.
- `session_duration_min`: duração média das sessões.
- `failed_logins_count`: tentativas falhas por janela de tempo.
- `2fa_opt_in_rate`: taxa de adesão ao duplo fator de autenticação.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Dispositivo] --> B[POST /auth/login]
    B --> C{Credenciais válidas?}
    C -- Não --> D[Erro 401 Unauthorized]
    C -- Sim --> E[Gerar user_id & session_id]
    E --> F[Retornar escopo autorizado]
```

✨ Fechamento da Camada

A **Camada 01** é o **portão de entrada** da IA Aurah Kosmos.

Cada usuário que entra no FriendApp é recebido com uma identidade única, **blindada por segurança técnica e governança de privacidade**, permitindo que todos os módulos seguintes operem com confiança.

Sem essa camada, nenhuma conexão autêntica poderia existir.

📦 CAMADA 02 — Cadastro Consciente & Perfil Base

🎯 Objetivo da Camada

Registrar o usuário no FriendApp de forma **consciente, transparente e técnica**, capturando não só dados básicos, mas também **preferências, limites de privacidade e primeiros traços de perfil vibracional**.

Essa camada é fundamental porque ela define a **primeira impressão da IA Aurah Kosmos sobre o usuário** e alimenta diretamente módulos como **Teste de Personalidade, Feed Sensorial e Mapa de Frequência**.

🔑 Entradas

- `user_id` (UUID gerado na Camada 01).
- `onboarding_responses`: respostas às perguntas de cadastro.

- `privacy_prefs`: preferências explícitas de privacidade (opt-ins, opt-outs).
- `consent_version`: versão do termo aceito.
- `duc_dco_status`: status da verificação documental (quando aplicável).

📤 Saídas

- `profile_traits`: mapa de traços iniciais do usuário (`sociabilidade`, `preferencia_eventos`, `intencao_de_uso`).
- `privacy_settings`: configuração granular de dados.
- `onboarding_completion_time_ms`: tempo que o usuário levou para completar cadastro.
- `profile_completion_score ∈ [0,1]`: grau de completude do perfil inicial.

📊 Regras Técnicas

- **Fluxo simplificado:** máximo de 5 etapas para reduzir fricção.
- **Consentimento explícito:** cada uso de dado deve ter caixa de seleção clara.
- **Controle granular:** usuário pode escolher compartilhar ou não dados sensíveis (idade, gênero, localização precisa).
- **Fallback automático:** se informações não forem fornecidas, defaults neutros são usados (ex.: localização = cidade, não GPS).

🔧 Exemplo de Contrato de API

Requisição

`POST /profile/init`

```
{
  "user_id": "6d71b6c2-84df-4de2-bc1d-9a8ab22a87d3",
  "onboarding_responses": {
    "prefer_social": true,
    "prefer_events": ["música", "arte"],
    "prefer_privacy_level": "alta"
  },
  "privacy_prefs": {
```

```
    "share_location": false,
    "share_contacts": false},
  "consent_version": "1.3"
}
```

Resposta

```
{
  "profile_traits": {
    "sociabilidade": 0.7,
    "preferencia_eventos": ["música", "arte"],
    "intencao_de_uso": "amizades"
  },
  "profile_completion_score": 0.92,
  "onboarding_completion_time_ms": 183000
}
```

🧪 Métricas de Observabilidade

- `drop_off_rate`: taxa de abandono durante cadastro.
- `consent_reject_rate`: quantos recusaram certos opt-ins.
- `avg_completion_time_ms`: tempo médio de finalização.
- `profile_completion_distribution`: distribuição de perfis completos x incompletos.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Usuário inicia cadastro] --> B[Onboarding respostas]
    B --> C[Configuração de privacidade]
    C --> D{Consentimento aceito?}
    D -- Não --> E[Abortar cadastro]
    D -- Sim --> F[Gerar profile_traits + score]
    F --> G[Persistir no DB e liberar Camada 03]
```

✨ Fechamento da Camada

A **Camada 02** é o **primeiro espelho do usuário dentro do FriendApp**.

Ela garante que a entrada seja feita de forma **ética, consciente e respeitosa**, ao mesmo tempo que cria a base técnica (`profile_traits`) que alimentará todas as recomendações da Aurah Kosmos.

Sem essa camada, a IA não teria dados confiáveis para iniciar qualquer interação.

📦 CAMADA 03 — Teste de Personalidade (Embedding Pessoal)

🎯 Objetivo da Camada

Gerar um **vetor de personalidade único para cada usuário** (`personality_vector`) que representa sua energia, estilo de interação e predisposições sociais.

Esse vetor é a **base matemática da identidade vibracional do usuário**, alimentando diretamente o **match de conexões autênticas, curadoria do feed e itinerários no mapa de frequência**.

🔑 Entradas

- `user_id` : identificador único do usuário (Camada 01).
- `profile_traits` : mapa inicial gerado no cadastro (Camada 02).
- `quiz_responses` : respostas ao teste de personalidade.
- `response_time_ms` : tempo gasto em cada resposta (indicador de confiança/hesitação).
- `skip_count` : número de perguntas ignoradas ou puladas.

📤 Saídas

- `personality_vector: float[128]` : vetor numérico em espaço de 128 dimensões.
- `confidence_score ∈ [0,1]` : confiança da IA na acurácia do vetor (ajustado por `skip_count` e consistência).
- `dominant_traits` : lista dos 3 traços mais fortes detectados.

📊 Regras Técnicas

- **Mínimo de 20 perguntas**, cobrindo dimensões como: sociabilidade, empatia, resiliência, abertura cultural.
- **Normalização Z-score** aplicada sobre os resultados para reduzir viés cultural.
- **Tempo de resposta** influencia o peso: respostas rápidas reforçam assertividade; lentas elevam incerteza.
- **Fallback**: se o usuário não completar o teste, `personality_vector` é estimado via heurísticas de perfil (Camada 02) + comportamento inicial no app.

🔧 Exemplo de Contrato de API

Requisição

`POST /personality/score`

```
{
  "user_id": "6d71b6c2-84df-4de2-bc1d-9a8ab22a87d3",
  "quiz_responses": {
    "q1": 5,
    "q2": 2,
    "q3": 4
  },
  "response_time_ms": {
    "q1": 1200,
    "q2": 3400,
    "q3": 800
  },
  "skip_count": 1
}
```

Resposta

```
{
  "personality_vector": [0.14, -0.21, 0.87, ...],
  "confidence_score": 0.82,
```

```
  "dominant_traits": ["empatia", "sociabilidade", "curiosidade"]
}
```

🧪 Métricas de Observabilidade

- `avg_response_time_ms`: tempo médio por pergunta.
- `consistency_index ∈ [0,1]`: mede coerência interna das respostas.
- `confidence_distribution`: distribuição de scores de confiança entre usuários.
- `skip_rate`: proporção de perguntas ignoradas.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Usuário responde quiz] ⟶ B[Captura respostas + tempos]
    B ⟶ C[Normalização + análise de consistência]
    C ⟶ D[LLM scoring + embedding]
    D ⟶ E[Gerar personality_vector]
    E ⟶ F[Armazenar no banco + liberar Camada 04]
```

✨ Fechamento da Camada

A **Camada 03** é onde o **subjetivo vira matemático**: respostas, hesitações e escolhas são transformadas em um vetor técnico que a Aurah Kosmos pode interpretar.

Esse vetor não é fixo: ele pode ser **refinado ao longo do tempo**, mas sempre servirá como a **chave mestra** da identidade vibracional de cada usuário no FriendApp.

📦 CAMADA 04 — Estado Emocional Instantâneo (NLP + Signals)

🎯 Objetivo da Camada

Inferir o **estado emocional atual do usuário** em tempo real, combinando **análise de linguagem natural (NLP)** com **sinais comportamentais de uso**.

Esse score (`emotional_state_score`) serve como insumo para **feed, chat, recomendações de eventos e alertas de segurança**, permitindo que a IA Aurah Kosmos adapte sua atuação ao momento presente.

🔑 Entradas

- `user_id`: referência ao usuário ativo.
- `text_input`: mensagens enviadas no chat, posts, comentários.
- `media_meta`: metadados de conteúdos consumidos (ex.: vídeo assistido até o fim).
- `ui_signals`: cliques, rolagem, tempo de tela, abandono de tela.
- `session_activity`: número de ações por minuto.

📤 Saídas

- `emotional_state_score ∈ [0,1]`: score contínuo de estado emocional (0 = negativo, 1 = positivo).
- `valence ∈ [-1,1]`: polaridade emocional.
- `arousal ∈ [0,1]`: nível de ativação/excitação.
- `confidence_score ∈ [0,1]`: nível de confiança da IA na inferência.

📊 Regras Técnicas

- **Sentimento normalizado**: baseado em embeddings pré-treinados de sentimento (foundation models).
- **Atividade normalizada**: detecção de quedas bruscas em interação → possível retração emocional.
- **Média ponderada**: pesos ajustáveis entre texto (linguagem explícita) e sinais (comportamento implícito).
- **Fallback neutro**: se não houver dados recentes, retorna estado = 0.5 (neutro).

🔧 Fórmulas

1. **Sentimento normalizado**

```
sentiment_norm = (sent_positive - sent_negative + 1)/2
```

1. **Atividade normalizada**

```
activity_norm = clamp(zscore(actions_per_min), 0, 1)
```

1. **Score emocional final**

```
emotional_state_score = 0.6 * sentiment_norm + 0.4 * (1 - activity_drop)
```

🔧 Exemplo de Contrato de API

Requisição

`POST /emotion/ingest`

```
{
  "user_id": "6d71b6c2-84df-4de2-bc1d-9a8ab22a87d3",
  "text_input": "Não estou animado para sair hoje",
  "ui_signals": {
    "actions_per_min": 2,
    "last_click_latency_ms": 3200
  },
  "session_activity": 0.3
}
```

Resposta

```
{
  "emotional_state_score": 0.28,
  "valence": -0.6,
  "arousal": 0.2,
  "confidence_score": 0.87
```

```
}
```

🧪 Métricas de Observabilidade

- `avg_emotion_score`: média da distribuição populacional por dia.
- `emotion_volatility`: variação do score em janelas de 15 min.
- `false_positive_rate`: erros detectados via feedback do usuário.
- `signal_dropout_rate`: proporção de sessões sem sinais suficientes.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Captura Texto + Sinais] ⟶ B[NLP Sentiment Analysis]
    A ⟶ C[UI Telemetry Analysis]
    B ⟶ D[Score Ponderado]
    C ⟶ D
    D ⟶ E[emotional_state_score]
    E ⟶ F[Persistir + Liberar Camada 05]
```

✨ Fechamento da Camada

A **Camada 04** é o **pulso vivo** do FriendApp: converte sentimentos e microcomportamentos em dados objetivos que a IA pode interpretar.

Sem esse score, a Aurah Kosmos não teria como adaptar **o tom do chat, a prioridade do feed ou a proteção contra colapsos**.

É a camada que dá **consciência emocional em tempo real** à IA.

📦 CAMADA 05 — Hesitação & Atenção (UX Telemetry)

🎯 Objetivo da Camada

Medir e interpretar a **hesitação do usuário frente às interfaces do FriendApp**, traduzindo esse comportamento em uma métrica (`attention_hesitation_ms`).

Essa camada permite à IA Aurah Kosmos identificar momentos de **dúvida, indecisão ou falta de engajamento** e calibrar a experiência do app (UI/UX, feed e sugestões) para reduzir atrito e aumentar coerência.

🔑 Entradas

- `user_id`: identificador do usuário ativo.
- `cta_render_ts`: timestamp da renderização de um CTA (call to action).
- `cta_click_ts`: timestamp da interação ou clique do usuário.
- `cta_type`: tipo de CTA (ex.: “entrar no grupo Bora”, “confirmar check-in”).
- `baseline_hesitation_ms`: média histórica de latência por usuário (EWMA).

📤 Saídas

- `attention_hesitation_ms`: tempo exato entre exibição e ação.
- `hesitation_delta`: diferença entre ação atual e baseline.
- `hesitation_flag`: alerta binário (true/false) caso a hesitação seja anormal (>2σ do baseline).

📊 Regras Técnicas

- **Baseline individualizado**: cada usuário tem seu `baseline_hesitation_ms`.
- **Z-score aplicado**: padronização das variações para detectar desvios.
- **Eventos críticos**: CTAs de segurança e Premium têm thresholds mais rígidos.
- **Fallback**: se não há histórico, baseline = média populacional por tipo de CTA.

🔧 Fórmulas

1. **Cálculo de hesitação**

```
attention_hesitation_ms = cta_click_ts - cta_render_ts
```

1. **Delta em relação ao baseline**

```
hesitation_delta = attention_hesitation_ms - baseline_hesitation_ms
```

1. **Flag de anomalia**

```
hesitation_flag = (hesitation_delta > 2 * sd_baseline)
```

🔧 Exemplo de Contrato de API

Requisição

`POST /ux/hesitation`

```
{
  "user_id": "6d71b6c2-84df-4de2-bc1d-9a8ab22a87d3",
  "cta_type": "join_bora",
  "cta_render_ts": 1726251000,
  "cta_click_ts": 1726251200
}
```

Resposta

```
{
  "attention_hesitation_ms": 200,
  "hesitation_delta": -50,
  "hesitation_flag": false}
```

🧪 Métricas de Observabilidade

- `avg_hesitation_ms`: média de hesitação por tipo de CTA.
- `hesitation_anomaly_rate`: % de flags de hesitação anormal.

- `cta_completion_rate`: proporção de CTAs clicados vs ignorados.
- `user_variance_index`: dispersão do comportamento individual vs coletivo.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Renderização do CTA] ⟶ B[Captura timestamps]
    B ⟶ C[Calcula atenção_hesitation_ms]
    C ⟶ D[Compara com baseline individual]
    D ⟶ E{Delta > 2σ?}
    E -- Sim ⟶ F[hesitation_flag = true]
    E -- Não ⟶ G[hesitation_flag = false]
    F ⟶ H[Persistir + alertar IA Aurah]
    G ⟶ H
```

✨ Fechamento da Camada

A **Camada 05** é a lente que revela o **tempo de dúvida do usuário**.

Esse pequeno intervalo, invisível no uso normal, se torna dado concreto para a IA, ajudando a calibrar interações e **reduzir fricção na jornada**.

Aqui, até a hesitação vira insumo para criar experiências mais fluidas e respeitosas.

📦 CAMADA 06 — Compatibilidade & Afinidade (Embeddings)

🎯 Objetivo da Camada

Construir o **vetor de compatibilidade (`compatibility_vector`)** de cada usuário e calcular a afinidade entre pares, grupos ou experiências.

Essa camada é a base do **sistema de conexões autênticas**, permitindo que a Aurah Kosmos sugira pessoas, grupos e eventos de forma **matemática, justa e transparente**.

🔑 Entradas

- `user_id`: identificador único.
- `personality_vector`: embedding da Camada 03.
- `emotional_state_score`: estado emocional instantâneo (Camada 04).
- `profile_traits`: dados do cadastro (Camada 02).
- `interaction_history`: registros de chats, encontros, eventos participados.
- `context_vector`: embedding contextual (Camada 07).

📤 Saídas

- `compatibility_vector: float[256]`: vetor multidimensional representando a energia social do usuário.
- `match_score ∈ [0,1]`: afinidade entre dois usuários.
- `context_fit ∈ [0,1]`: ajuste ao contexto atual (evento, geo, horário).
- `history_overlap ∈ [0,1]`: grau de sobreposição de histórico.

📊 Regras Técnicas

- **Embeddings concatenados**: união de `personality_vector`, `context_vector` e histórico normalizado.
- **Normalização L2**: todos os vetores ajustados para módulo = 1.
- **Atualização incremental**: compatibilidade recalculada a cada nova interação relevante.

🔧 Fórmulas

1. **Similaridade vetorial (cosine similarity)**

```
sim(A,B) = cosine(compatibility_vector_A, compatibility_vector_B)
```

1. **Score de afinidade final**
    - **Redução de dimensionalidade**: PCA ou UMAP aplicados para performance.

```
match_score = 0.5 * sim(A,B)
            + 0.3 * history_overlap(A,B)
            + 0.2 * context_fit(A,B)
```

🔧 Exemplo de Contrato de API

Requisição

`POST /match/compute`

```
{
  "user_a": "uuid_A",
  "user_b": "uuid_B"
}
```

Resposta

```
{
  "match_score": 0.82,
  "context_fit": 0.75,
  "history_overlap": 0.6,
  "explanation": "Alta afinidade baseada em interesses comuns e histórico de interações"
}
```

🧪 Métricas de Observabilidade

- `avg_match_score`: média populacional de matches.
- `distribution_curve`: curva de afinidades por cluster.
- `false_positive_rate`: usuários que rejeitam conexões sugeridas com score > 0.7.
- `update_latency_ms`: tempo de atualização do vetor de compatibilidade.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Captura perfil + histórico + contexto] --> B[Normalização e concatenação]
    B --> C[Redução dimensional]
    C --> D[Gerar compatibility_vector]
    D --> E[Comparação entre pares]
    E --> F[match_score]
    F --> G[Persistir no banco + liberar Camada 07]
```

✨ Fechamento da Camada

A **Camada 06** traduz em matemática aquilo que, no mundo real, chamamos de "química" ou "compatibilidade".

Cada sugestão de conexão no FriendApp é baseada em embeddings e fórmulas transparentes, evitando vieses ocultos e garantindo que o usuário entenda **por que** recebeu aquela sugestão.

Aqui, afinidade é ciência, não acaso.

📦 CAMADA 07 — Contexto Vivo (Geo/Tempo/Evento)

🎯 Objetivo da Camada

Construir o **vetor de contexto (** `context_vector` **)** que representa o ambiente em que o usuário está inserido em cada sessão:

- **Onde** ele está (geo),
- **Quando** ele está ativo (tempo),
- **O que** acontece ao redor (eventos).

Esse vetor permite à Aurah Kosmos conectar o perfil e estado emocional do usuário ao **mundo real**, ajustando recomendações, matches e itinerários no mapa de frequência.

🔑 Entradas

- `user_id`: identificador do usuário ativo.
- `geo_hash`: localização aproximada (5–7 caracteres, precisão de bairro/região).
- `local_time`: timestamp da sessão.
- `event_tags`: eventos ativos próximos (ex.: música, networking, mindfulness).
- `device_context`: metadados do dispositivo (wifi, bateria, modo noturno).

📤 Saídas

- `context_vector: float[64]`: embedding numérico representando o contexto.
- `context_fit ∈ [0,1]`: grau de adequação entre contexto atual e sugestão (eventos, conexões).
- `geo_cluster_id`: cluster geográfico do usuário.
- `time_bucket`: bucket de tempo (ex.: manhã, tarde, noite).

📊 Regras Técnicas

- **Geoprivacidade**: localização só é salva como `geo_hash`, nunca como lat/long precisa.
- **Contexto temporal**: uso de buckets horários e sazonais para capturar padrões (ex.: finais de semana sociais).
- **Eventos integrados**: ingestão via API de locais parceiros + curadoria manual.
- **Atualização em tempo real**: vetor recalculado a cada 15 min ou mudança de localização.

🔧 Fórmulas

1. **Normalização temporal**

```
time_bucket = floor((local_time % 86400) / 14400)  // 6 buckets de 4h
```

1. **Score de contexto para evento**

```
context_fit = 0.5 * geo_similarity(user, event)
            + 0.3 * time_overlap(user, event)
            + 0.2 * tag_match(user, event)
```

🔧 Exemplo de Contrato de API

Requisição

`POST /context/update`

```
{
  "user_id": "6d71b6c2-84df-4de2-bc1d-9a8ab22a87d3",
  "geo_hash": "6gkz7",
  "local_time": 1726354800,
  "event_tags": ["música", "arte"]
}
```

Resposta

```
{
  "context_vector": [0.23, -0.11, 0.84, ...],
  "context_fit": 0.77,
  "geo_cluster_id": "cluster_112",
  "time_bucket": "noite"
}
```

🧪 Métricas de Observabilidade

- `avg_context_fit`: média de adequação contexto-ação.
- `geo_entropy`: diversidade de geohashes por usuário.
- `event_tag_match_rate`: % de tags de eventos consumidos vs sugeridos.
- `context_update_latency_ms`: tempo para recalcular vetor de contexto.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Captura geo + tempo + eventos] --> B[Normalização]
    B --> C[Construção context_vector]
    C --> D[Calcular context_fit p/ eventos ativos]
    D --> E[Persistir contexto no banco]
    E --> F[Liberar para Camada 08 - Curadoria de Conteúdo]
```

✨ Fechamento da Camada

A **Camada 07** é o **elo entre o digital e o físico**.

Ela garante que o FriendApp não seja apenas um app de tela, mas um **espelho do ambiente real** do usuário, permitindo que a IA Aurah Kosmos sugira experiências que fazem sentido **no lugar, no momento e no clima certos**.

📦 CAMADA 08 — Curadoria de Conteúdo (Feed)

🎯 Objetivo da Camada

Selecionar, priorizar e exibir conteúdos no feed do usuário com base em **relevância emocional, contexto e impacto positivo esperado**.

Essa camada garante que o feed do FriendApp não seja uma timeline aleatória, mas sim um **espaço vivo de conexões reais**, balanceando exploração de novidades com segurança vibracional.

🔑 Entradas

- `user_id`: identificador único.
- `personality_vector`: embedding da Camada 03.
- `emotional_state_score`: estado atual do usuário (Camada 04).
- `compatibility_vector`: embedding social (Camada 06).
- `context_vector`: ambiente atual (Camada 07).

- `post_metadata`: informações do conteúdo (autor, tags, tipo de mídia).
- `trust_score`: reputação técnica/social do usuário (Camada 35).

📤 Saídas

- `feed_rank`: score de priorização por conteúdo.
- `feed_items`: lista de posts recomendados ordenados por rank.
- `why_explanation`: explicação curta para cada recomendação ("Sugerido porque você curte música + alta afinidade com João").

📊 Regras Técnicas

- **Exploração vs Exploitation**: balanceamento via algoritmo *ε-greedy*.
- **Penalização de risco**: posts marcados como negativos/denunciados têm peso reduzido.
- **Boost de afinidade**: conteúdos de conexões autênticas têm prioridade.
- **Exploração controlada**: posts fora do perfil exibidos em <15% do feed.
- **Feedback em loop**: cada interação (curtir, esconder, denunciar) recalibra os modelos.

🔧 Fórmulas

1. **Rank de feed básico**

```
feed_rank = 0.5 * relevance
          + 0.35 * expected_uplift
          - 0.15 * risk
```

1. **Probabilidade de exploração**

```
P(exploração) = ε / |C|
```

onde `ε` = 0.1 (10% do feed), `|C|` = número de candidatos.

🔧 Exemplo de Contrato de API

Requisição

`GET /feed?user_id=6d71b6c2-84df-4de2-bc1d-9a8ab22a87d3`

Resposta


```
[
  {
    "post_id": "123",
    "feed_rank": 0.87,
    "why_explanation": "Alta afinidade com perfil de Maria + interesse em ev
entos de música"
  },
  {
    "post_id": "456",
    "feed_rank": 0.65,
    "why_explanation": "Exploração controlada: evento de mindfulness próxi
mo"
  }
]
```

🧪 Métricas de Observabilidade

- `ctr_feed`: taxa de cliques em posts exibidos.
- `avg_feed_rank_interacted`: média dos ranks dos posts clicados.
- `hide_rate`: % de posts ocultados.
- `exploration_acceptance`: taxa de engajamento em posts exploratórios.
- `uplift_realized`: diferença média de `emotional_state_score` após consumir conteúdo.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Posts candidatos] --> B[Filtragem básica]
    B --> C[Calcular relevance/expected_uplift/risk]
    C --> D[Algoritmo ε-greedy]
```

```
D --> E[Rank final + seleção top-N]
E --> F[Exibir feed ordenado + why_explanation]
F --> G[Captura feedback (curtir, esconder, denunciar)]
G --> H[Atualiza modelos e scores]
```

✨ Fechamento da Camada

A **Camada 08** é onde o FriendApp mostra seu diferencial:

o feed não é uma corrida por atenção, mas um **espaço intencional** onde cada post é exibido porque faz sentido para aquele usuário, naquele momento.

Com explicações claras (“por que estou vendo isso?”), o feed se torna um **instrumento de confiança e evolução**, não de manipulação.

📦 CAMADA 09 — Conexões Autênticas (Matching Social)

🎯 Objetivo da Camada

Gerar sugestões de **amizades, grupos e conexões reais** no FriendApp a partir da afinidade entre usuários.

Essa camada traduz os **vetores de compatibilidade** (Camada 06) em **ações sociais concretas**, priorizando qualidade, segurança e autenticidade acima de quantidade.

🔑 Entradas

- `user_id`: identificador do usuário ativo.
- `compatibility_vector`: embedding calculado na Camada 06.
- `match_score`: afinidade entre usuários.
- `context_fit`: adequação contextual do momento (Camada 07).
- `history_overlap`: sobreposição de interações anteriores.
- `safety_flags`: bloqueios, denúncias, reputação de ambos.

📤 Saídas

- `connection_priority`: ranking de prioridade da sugestão.
- `suggested_peers`: lista de usuários sugeridos, com explicação.
- `group_recommendations`: grupos Bora/temáticos sugeridos.
- `why_explanation`: justificativa clara para cada sugestão.

📊 Regras Técnicas

- **Safety first**: bloqueios e denúncias zeram a prioridade de conexão.
- **Transparência**: cada sugestão deve vir acompanhada de explicação curta.
- **Diversidade controlada**: sugestões não podem ser todas iguais (mesma faixa etária ou mesmo cluster).
- **Histórico ajustado**: afinidades reforçadas por eventos ou interações recentes ganham peso extra.

🔧 Fórmula

```
connection_priority = match_score * intent_alignment * safety_gate
```

- `intent_alignment ∈ [0,1]`: alinhamento entre intenções declaradas no perfil (amizade, networking, hobby).
- `safety_gate`: binário → 0 se houver bloqueio/denúncia ativa.

🔧 Exemplo de Contrato de API

Requisição

`POST /connections/suggest`

```
{
  "user_id": "uuid_A"
}
```

Resposta

```
{
  "suggested_peers": [
    {
      "peer_id": "uuid_B",
      "connection_priority": 0.83,
      "why_explanation": "Vocês compartilham interesse em música + partici
param de eventos semelhantes"
    },
    {
      "peer_id": "uuid_C",
      "connection_priority": 0.72,
      "why_explanation": "Compatibilidade alta no perfil de personalidade e p
roximidade geográfica"
    }
  ],
  "group_recommendations": [
    {
      "group_id": "g123",
      "topic": "Arte & Criatividade",
      "why_explanation": "Alta afinidade com seus interesses culturais"
    }
  ]
}
```


🧪 Métricas de Observabilidade

- `acceptance_rate`: % de sugestões aceitas.
- `false_positive_rate`: % de sugestões rejeitadas com match_score > 0.7.
- `connection_priority_distribution`: curva de ranks sugeridos.
- `safety_gate_trigger_rate`: % de conexões barradas por risco.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Captura match_score + context_fit] --> B[Aplica safety_gate]
```

```
B --> C[Calcula connection_priority]
C --> D[Rankeia peers e grupos]
D --> E[Exibe sugestões + explicações]
E --> F[Captura feedback do usuário]
F --> G[Recalibra modelos de afinidade]
```

✨ Fechamento da Camada

A **Camada 09** é onde o FriendApp cumpre sua promessa central:

transformar dados em conexões humanas reais.

Aqui, afinidade não é só cálculo: é um convite claro, seguro e transparente para o usuário expandir seu círculo de pertencimento.

📦 CAMADA 10 — Locais & Eventos (Recomendação Contextual)

🎯 Objetivo da Camada

Recomendar **locais parceiros e eventos sociais** que façam sentido para o usuário no seu contexto atual, conectando o digital ao físico.

Essa camada transforma **o mapa de frequência e o vetor de contexto** em **convites reais**, criando pontes entre a IA Aurah Kosmos e experiências concretas.

🔑 Entradas

- `user_id`: identificador único do usuário.
- `context_vector`: vetor de contexto vivo (Camada 07).
- `compatibility_vector`: embedding social (Camada 06).
- `event_metadata`: informações sobre eventos ativos (nome, local, horário, tags).
- `partner_location_data`: informações de locais parceiros (geo, categoria, reputação).
- `expected_uplift`: ganho esperado no `emotional_state_score` ao participar.

📤 Saídas

- `event_fit ∈ [0,1]`: adequação do evento ao usuário.
- `location_recommendations`: lista de locais sugeridos.
- `event_recommendations`: lista de eventos priorizados.
- `why_explanation`: justificativa clara (“Sugerido porque você gosta de arte + proximidade geográfica”).

📊 Regras Técnicas

- **Check-ins obrigatórios**: participação em evento/estabelecimento deve ser registrada para aprendizado da IA.
- **Segurança vibracional**: eventos em locais com baixa reputação são rebaixados no ranking.
- **Diversidade contextual**: não sugerir sempre o mesmo tipo de evento.
- **Transparência**: cada sugestão deve vir com explicação legível.

🔧 Fórmula

```
event_fit = 0.6 * context_fit
          + 0.4 * expected_uplift
```

🔧 Exemplo de Contrato de API

Requisição

`POST /events/recommend`

```
{
  "user_id": "6d71b6c2-84df-4de2-bc1d-9a8ab22a87d3",
  "context_vector": [0.23, -0.11, 0.84, ...],
  "event_metadata": [
    {"event_id": "e123", "tags": ["arte", "música"], "geo_hash": "6gkz7"}
  ]
```

```
}
```

Resposta

```
{
  "event_recommendations": [
    {
      "event_id": "e123",
      "event_fit": 0.81,
      "why_explanation": "Evento de arte próximo + afinidade com seus inter
esses culturais"
    }
  ],
  "location_recommendations": [
    {
      "partner_id": "p456",
      "score": 0.74,
      "why_explanation": "Local parceiro de música, compatível com seu perf
il social"
    }
  ]
}
```

🧪 Métricas de Observabilidade

- `checkin_rate`: % de eventos sugeridos com check-in realizado.
- `avg_event_fit`: média de adequação evento-usuário.
- `drop_off_after_suggestion`: usuários que receberam convite mas não compareceram.
- `partner_engagement_score`: média de engajamento em locais parceiros.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Contexto vivo + compatibilidade] --> B[Filtrar eventos/locais ativos]
```

```
B --> C[Calcular event_fit]
C --> D[Rankear recomendações]
D --> E[Exibir lista + why_explanation]
E --> F[Captura check-in e feedback]
F --> G[Recalibra modelos e expected_uplift]
```

✨ Fechamento da Camada

A **Camada 10** é onde o FriendApp deixa de ser apenas digital:

ela leva o usuário para **experiências reais**, em locais e eventos que fazem sentido para sua energia e momento.

Aqui, a IA não sugere apenas um post — ela sugere **um lugar para estar, uma experiência para viver e uma memória para criar**.

📦 CAMADA 11 — Chat Tático (Tom & Ritmo)

🎯 Objetivo da Camada

Ajustar o **tom, o ritmo e o conteúdo das respostas no chat** de acordo com o estado emocional do usuário (`emotional_state_score`) e o contexto.

Essa camada garante que a IA Aurah Kosmos se comunique de forma **humana, respeitosa e proporcional**, seguindo o princípio da **subsidiariedade**: agir menos, sugerir mais, sempre deixando o usuário no controle.

🔑 Entradas

- `user_id`: identificador do usuário ativo.
- `session_context`: dados da sessão (geo, tempo, interação atual).
- `emotional_state_score`: estado emocional (Camada 04).
- `compatibility_vector`: afinidade social (Camada 06).
- `chat_history`: histórico de conversas.
- `safety_flags`: moderador de risco (Camada 15).

📤 Saídas

- `reply`: resposta sugerida para o usuário.
- `tone_label`: rótulo do tom (ex.: empático, neutro, animado, calmo).
- `rhythm_label`: ajuste de ritmo (curto, médio, longo).
- `risk_flag`: alerta binário (true/false) em caso de risco detectado.
- `why_explanation`: justificativa técnica do ajuste.

📊 Regras Técnicas

- **Tom proporcional**:
  - Estado positivo → tom mais leve e expansivo.
  - Estado neutro → tom neutro, objetivo.
  - Estado negativo → tom calmo, empático.
- **Controle de ritmo**:
  - Usuários hesitantes → respostas curtas e claras.
  - Usuários engajados → respostas mais longas e ricas.
- **Segurança integrada**:
  - Palavras de risco (autoagressão, assédio) → `risk_flag = true` e ativação da Camada 15.
- **Explicabilidade**: cada resposta deve ser acompanhada de metadado com o motivo do tom escolhido.

🔧 Fórmula Simplificada

```
tone_label = f(emotional_state_score, safety_flags, chat_history)
rhythm_label = g(hesitation_delta, engagement_level)
```

🔧 Exemplo de Contrato de API

Requisição

`POST /chat/reply`

```
{
  "user_id": "6d71b6c2-84df-4de2-bc1d-9a8ab22a87d3",
  "session_context": {"geo": "6gkz7", "time_bucket": "noite"},
  "emotional_state_score": 0.32,
  "chat_history": ["Hoje foi um dia difícil..."]
}
```

Resposta

```
{
  "reply": "Entendo, dias assim podem ser pesados. Quer compartilhar um pouco mais?",
  "tone_label": "empático",
  "rhythm_label": "curto",
  "risk_flag": false,
  "why_explanation": "Score emocional baixo + linguagem negativa → resposta curta e acolhedora"
}
```

🧪 Métricas de Observabilidade

- `avg_reply_latency_ms`: tempo médio para resposta.
- `tone_distribution`: proporção de tons usados (empático, neutro, animado).
- `false_positive_risk_flags`: respostas marcadas como risco indevidamente.
- `user_feedback_useful`: % de respostas avaliadas como úteis pelo usuário.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Captura estado emocional + histórico] --> B[Classificação de tom]
    A --> C[Análise de ritmo e hesitação]
    B --> D[Gerar reply]
    C --> D
    D --> E[Adicionar metadados de explicação]
```

```
    E --> F[Exibir resposta ao usuário]
    F --> G[Capturar feedback (foi útil?)]
    G --> H[Ajustar modelos de tom e ritmo]
```

✨ Fechamento da Camada

A **Camada 11** é onde o chat do FriendApp deixa de ser mecânico e se torna **tático e humano**.

A IA Aurah Kosmos não apenas responde, mas responde **no tom certo, com o ritmo certo, no momento certo** — sempre de forma clara, ética e transparente.

Aqui, o usuário se sente compreendido, nunca manipulado.

📦 CAMADA 12 — Mapa de Estado Coletivo (Aggregate)

🎯 Objetivo da Camada

Construir uma visão **agregada do estado emocional coletivo** em diferentes regiões geográficas, transformando dados individuais em um **heatmap vivo**.

Essa camada permite à IA Aurah Kosmos identificar **zonas de expansão, retração ou colapso coletivo**, ajudando a calibrar recomendações, eventos e até missões regenerativas.

🔑 Entradas

- `user_id`: identificador anônimo (não PII).
- `geo_hash`: localização aproximada do usuário (5–7 caracteres).
- `emotional_state_score`: estado individual (Camada 04).
- `context_vector`: informações temporais e ambientais (Camada 07).
- `checkin_events`: registros de presença em eventos/locais (Camada 10).

📤 Saídas

- `aggregate_emotional_state ∈ [0,1]`: score médio da região.
- `heatmap_cells`: células geográficas com status (expansão, retração, neutra).
- `collective_trend`: tendência de evolução do estado coletivo.
- `alerts`: disparo de alertas para IA em casos de colapso coletivo.

📊 Regras Técnicas

- **Agregação ponderada**: mais peso para usuários ativos e recentes.
- **Anonimização**: todos os dados são agregados; nenhuma célula pode ter < 20 usuários ativos.
- **Normalização temporal**: scores são suavizados por janelas de 15 min.
- **Alertas condicionais**: colapsos regionais só disparam alertas se persistirem por >30 min.

🔧 Fórmula

```
aggregate_emotional_state_cell =
  Σ (emotional_state_score_i * activity_weight_i) / Σ activity_weight_i
```

- `activity_weight_i`: peso baseado na recência e engajamento do usuário.

🔧 Exemplo de Contrato de API

Requisição

`GET /map/collective?geo_hash=6gkz7`

Resposta

```
{
  "geo_hash": "6gkz7",
  "aggregate_emotional_state": 0.62,
  "collective_trend": "expansao",
  "heatmap_cells": [
    {"cell_id": "6gkz7", "state": "expansao", "score": 0.62},
    {"cell_id": "6gkz8", "state": "neutro", "score": 0.51}
  ],
```

```
  "alerts": []
}
```

🧪 Métricas de Observabilidade

- `avg_aggregate_score`: média global do estado coletivo.
- `expansion_zone_count`: número de zonas em expansão.
- `collapse_alerts_triggered`: quantidade de colapsos detectados.
- `user_to_cell_ratio`: proporção de usuários por célula válida.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Captura emotional_state_scores individuais] --> B[Agrupa por geo_has
h]
    B --> C[Pondera por atividade/recência]
    C --> D[Calcula média agregada da célula]
    D --> E[Atualiza heatmap]
    E --> F{Persistência do padrão > 30min?}
    F -- Sim --> G[Dispara alerta coletivo]
    F -- Não --> H[Manter status neutro]
```

✨ Fechamento da Camada

A **Camada 12** é o radar coletivo do FriendApp.

Ela transforma dados individuais em um **mapa vivo das emoções coletivas**, permitindo que a Aurah Kosmos entenda **o clima social de cada região** e atue com inteligência contextual.

É aqui que o app ganha a capacidade de enxergar **o coletivo como organismo vivo**.

📦 CAMADA 13 — Jogo de Evolução (Gamificação Ética)

🎯 Objetivo da Camada

Transformar a jornada do usuário em uma experiência de **crescimento contínuo** através de gamificação ética, que **recompensa ações saudáveis e pró-sociais** dentro e fora do app.

O objetivo não é viciar, mas sim **estimular evolução pessoal e conexões reais** de forma lúdica e transparente.

🔑 Entradas

- `user_id`: identificador único.
- `checkin_events`: presença em eventos/locais (Camada 10).
- `connection_actions`: novos matches, mensagens trocadas, amizades formadas.
- `feedback_quality`: avaliações dadas/recebidas.
- `mission_completion`: status de missões regenerativas (Camada 59).

📤 Saídas

- `xp_gain`: pontos de evolução obtidos.
- `level`: nível atual do usuário.
- `badges`: selos conquistados (amizade autêntica, impacto social, presença em eventos).
- `progress_bar`: feedback visual do progresso.

📊 Regras Técnicas

- **Reforço positivo**: ações pró-sociais geram XP.
- **Anti-gamificação**: bloqueio contra farming (ex.: enviar 100 mensagens vazias não gera pontos).
- **Equidade**: recompensas são qualitativas, não competitivas.
- **Transparência**: usuário vê sempre o porquê de cada ganho.

🔧 Fórmula de XP (exemplo)

```
xp_gain = 2 * checkin_events
        + 3 * connections_accepted
        + 4 * feedback_quality_score
        + 5 * mission_completion
```

- `checkin_events`: peso menor (ações simples).
- `connections_accepted`: peso médio (reforça vínculos).
- `feedback_quality_score`: avalia autenticidade e utilidade.
- `mission_completion`: maior peso (evolução interna).

🔧 Exemplo de Contrato de API

Requisição

`POST /game/progress`

```
{
  "user_id": "uuid",
  "checkin_events": 2,
  "connections_accepted": 1,
  "feedback_quality_score": 0.9,
  "mission_completion": 1
}
```

Resposta

```
{
  "xp_gain": 17,
  "level": 3,
  "badges": ["Conexão Autêntica", "Impacto Social"],
  "progress_bar": 0.68
}
```

🧪 Métricas de Observabilidade

- `xp_distribution`: distribuição de XP entre usuários.
- `mission_completion_rate`: % de missões completadas.
- `false_positive_xp`: XP obtido de forma indevida (fraude/farming).
- `badge_claim_rate`: proporção de selos efetivamente exibidos.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Captura ações do usuário] --> B[Valida se são pró-sociais]
    B --> C[Calcula xp_gain]
    C --> D[Atualiza nível e progresso]
    D --> E[Libera badges e recompensas]
    E --> F[Exibe evolução no painel do usuário]
```

✨ Fechamento da Camada

A **Camada 13** transforma cada interação em uma oportunidade de evolução.

Aqui, **conectar-se, ajudar, participar e cuidar** são recompensados com progresso visível.

É a forma da IA Aurah Kosmos mostrar que **crescer socialmente e emocionalmente é um jogo em que todos podem ganhar**.

📦 CAMADA 14 — Premium & FriendCoins (Economia)

🎯 Objetivo da Camada

Gerenciar o **sistema econômico do FriendApp**, combinando assinatura **Premium** e a moeda virtual interna **FriendCoins**.

O objetivo é criar uma **economia justa e transparente**, que sustente o app sem comprometer ética, segurança ou autenticidade das conexões.

🔑 Entradas

- `user_id`: identificador do usuário.
- `subscription_status`: estado do plano (Free, Premium Mensal, Premium Anual).
- `friendcoins_balance`: saldo atual de FriendCoins.
- `purchase_events`: compras realizadas (ex.: boost de evento, selo especial).
- `partner_transactions`: pagamentos de locais parceiros.

📤 Saídas

- `feature_access`: lista de funcionalidades habilitadas.
- `friendcoins_balance_updated`: saldo atualizado.
- `transaction_logs`: histórico de movimentações.
- `billing_status`: status de pagamento e renovação.

📊 Regras Técnicas

- **Gatilhos Premium**: criar eventos, ver Painel Vibracional e impulsionar posts → exclusivo Premium.
- **FriendCoins**: moeda interna que pode ser usada para impulsionar eventos, adquirir boosts ou selos especiais.
- **Transparência absoluta**: cada transação gera log com `timestamp`, `amount`, `reason`.
- **Anti-abuso**: sem pay-to-win → economia não afeta autenticidade, apenas acelera visibilidade.

🔧 Exemplo de Contrato de API

Requisição

`POST /billing/subscribe`

```
{
  "user_id": "uuid",
  "plan": "premium_mensal",
  "payment_token": "tok_visa_12345"
```

```
}
```

Resposta

```
{
  "subscription_status": "premium_mensal",
  "feature_access": ["criar_eventos", "painel_vibracional", "impulsionar_post
s"],
  "billing_status": "ativo"
}
```

Requisição (FriendCoins)

`POST /coins/use`

```
{
  "user_id": "uuid",
  "amount": 50,
  "reason": "boost_evento"
}
```

Resposta

```
{
  "friendcoins_balance_updated": 150,
  "transaction_logs": [
    {"amount": -50, "reason": "boost_evento", "timestamp": 1726400000}
  ]
}
```

🧪 Métricas de Observabilidade

- `premium_conversion_rate`: % de usuários que viram Premium.
- `avg_friendcoins_balance`: saldo médio por usuário.

- `transaction_volume`: volume diário de moedas transacionadas.
- `refund_rate`: taxa de pedidos de reembolso.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Usuário Free] --> B[Assina Premium?]
    B -- Sim --> C[Ativa features Premium]
    B -- Não --> D[Permanece Free]
    C --> E[Usa FriendCoins para boosts]
    D --> E
    E --> F[Atualiza saldo e gera logs]
```

✨ Fechamento da Camada

A **Camada 14** é a base da sustentabilidade econômica do FriendApp.

Aqui, **a monetização existe sem corromper a missão**: não há compra de conexões nem manipulação vibracional.

O Premium e os FriendCoins servem para **expandir experiências**, manter a plataforma saudável e financiar impacto social.

📦 CAMADA 15 — Segurança & Moderação (Firewall)

🎯 Objetivo da Camada

Detectar, prevenir e moderar **conteúdos e comportamentos nocivos** dentro do FriendApp, garantindo um ambiente seguro e acolhedor.

Essa camada funciona como o **firewall vibracional** da plataforma, cruzando inteligência artificial e curadoria humana para proteger usuários sem comprometer a experiência.

🔑 Entradas

- `user_id`: identificador único.
- `chat_messages`: mensagens trocadas em chats.
- `feed_posts`: conteúdos publicados no feed.
- `event_interactions`: interações em eventos e check-ins.
- `report_flags`: denúncias de usuários.
- `risk_indicators`: sinais de risco (linguagem tóxica, assédio, autoagressão).

📤 Saídas

- `risk_label`: classificação do risco (baixo, médio, alto).
- `actions`: medidas aplicadas (ex.: alerta, bloqueio temporário, revisão manual).
- `moderation_logs`: registros técnicos do processo de moderação.
- `escalation_flags`: sinalização para conselho humano em casos críticos.

📊 Regras Técnicas

- **Classificação automática**: modelos de NLP e detecção de imagem analisam linguagem, tom e contexto.
- **Thresholds dinâmicos**: calibrados para reduzir falsos positivos.
- **Escalonamento humano**: todo caso grave vai para revisão manual.
- **Transparência ao usuário**: toda ação gera explicação clara (“sua postagem foi removida por conter linguagem de ódio”).
- **Registro imutável**: logs de moderação são armazenados com hash para auditoria.

🔧 Fórmula de Score de Risco

```
risk_score = 0.4 * toxicity_score
           + 0.3 * harassment_score
           + 0.2 * self_harm_score
           + 0.1 * report_density
```

- Se `risk_score >= 0.8` → ação automática de bloqueio.
- Se `0.5 <= risk_score < 0.8` → alerta ao usuário + revisão manual.
- Se `< 0.5` → monitoramento passivo.

🔧 Exemplo de Contrato de API

Requisição

`POST /safety/moderate`

```
{
  "user_id": "uuid",
  "content": "Você não vale nada, deveria desistir",
  "reports": 3
}
```

Resposta

```
{
  "risk_label": "alto",
  "risk_score": 0.86,
  "actions": ["bloqueio_temporario", "escalonamento_humano"],
  "moderation_logs": [
    {"timestamp": 1726450000, "reason": "autoagressao", "hash": "abc123"}
  ]
}
```

🧪 Métricas de Observabilidade

- `false_positive_rate`: % de casos moderados indevidamente.
- `escalation_rate`: % de casos enviados para revisão humana.
- `avg_moderation_latency_ms`: tempo médio entre detecção e ação.
- `report_to_action_ratio`: proporção de denúncias que resultam em ação real.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Captura mensagens, posts, interações] --> B[Análise IA (toxicity, haras
sment, self-harm)]
    B --> C[Calcular risk_score]
    C --> D{risk_score >= threshold?}
    D -- Alto --> E[Ação automática + bloqueio]
    D -- Médio --> F[Alerta + revisão humana]
    D -- Baixo --> G[Monitoramento passivo]
    E --> H[Gerar moderation_logs]
    F --> H
    G --> H
```

✨ Fechamento da Camada

A **Camada 15** é o escudo invisível do FriendApp.

Ela garante que a plataforma seja um ambiente **seguro, ético e transparente**, onde cada usuário saiba que está protegido contra abusos.

Aqui, a moderação não é censura oculta: é **clareza, auditoria e responsabilidade**.

📦 CAMADA 16 — Transparência & Controles do Usuário

🎯 Objetivo da Camada

Colocar o **usuário no centro das decisões**, garantindo total transparência sobre como a IA Aurah Kosmos funciona e fornecendo controles claros para ajustar, limitar ou desligar recursos.

Essa camada é fundamental para criar **confiança**, mostrar que o FriendApp não manipula, e que cada pessoa tem o **poder final sobre seus dados e experiência**.

🔑 Entradas

- `user_id`: identificador único.
- `privacy_prefs`: preferências de privacidade definidas no cadastro (Camada 02).
- `feedback_signal`: feedback explícito do usuário (“isso ajudou?”, “ver menos disso”).
- `consent_flags`: consentimentos ativos (uso de localização, check-ins, IA proativa).
- `feature_toggle`: switches para ligar/desligar features.

📤 Saídas

- `updated_privacy_settings`: configurações aplicadas imediatamente.
- `feedback_logs`: registro do feedback coletado.
- `consent_audit_trail`: trilha de auditoria de consentimentos (com timestamps).
- `feature_status`: status atual de recursos ativos/desativados.

📊 Regras Técnicas

- **Feedback contínuo**: cada insight ou recomendação deve ter botões “foi útil?” e “ver menos disso”.
- **Opt-in/opt-out granular**: localização precisa, IA proativa, notificações → todos configuráveis individualmente.
- **Efeito imediato**: ao desligar um recurso, ele deve ser desativado em tempo real.
- **Auditoria completa**: logs de consentimento devem ser imutáveis e acessíveis ao usuário.

🔧 Exemplo de Contrato de API

Requisição (feedback)

`POST /user/feedback`

```
{
  "user_id": "uuid",
  "content_id": "post_123",
```

```
  "feedback_signal": "not_useful"
}
```

Resposta

```
{
  "status": "ok",
  "feedback_logs": [
    {"content_id": "post_123", "signal": "not_useful", "timestamp": 17264680
00}
  ]
}
```

Requisição (alterar preferências)

POST /user/privacy

```
{
  "user_id": "uuid",
  "share_location": false,
  "ai_proactivity": false}
```

Resposta

```
{
  "updated_privacy_settings": {
    "share_location": false,
    "ai_proactivity": false},
  "consent_audit_trail": [
    {"flag": "ai_proactivity", "status": "false", "timestamp": 1726468200}
  ]
}
```

🧪 Métricas de Observabilidade

- `feedback_util_rate`: % de feedbacks positivos.
- `opt_out_rate`: % de usuários que desativam recursos proativos.
- `audit_access_count`: quantas vezes usuários consultaram seus logs de consentimento.
- `avg_toggle_latency_ms`: tempo médio entre ajuste e efeito aplicado.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Usuário interage com feed/IA] --> B[Exibe botões de feedback]
    B --> C[Captura feedback_signal]
    C --> D[Atualiza modelos + logs]
    A --> E[Usuário ajusta preferências de privacidade]
    E --> F[Atualiza status em tempo real]
    F --> G[Registra em consent_audit_trail]
```

✨ Fechamento da Camada

A **Camada 16** garante que o FriendApp seja **um ecossistema de confiança**.

O usuário sabe exatamente **por que vê algo**, pode dizer quando não faz sentido, e tem controle total para limitar ou desligar recursos.

Aqui, a Aurah Kosmos não é uma força oculta: ela é **uma parceira transparente, sempre sob escolha do usuário**.

📦 CAMADA 17 — Personalização Cultural & Idioma

🎯 Objetivo da Camada

Adaptar a experiência do FriendApp à **cultura, idioma e contexto local** do usuário, garantindo que a IA Aurah Kosmos se comunique com **respeito, empatia e coerência cultural**.

Essa camada assegura que as interações não apenas sejam traduzidas, mas também **interpretadas** dentro da realidade sociocultural do usuário.

🔑 Entradas

- `user_id`: identificador único.
- `preferred_language`: idioma preferido definido no perfil.
- `geo_hash`: região aproximada (Camada 07).
- `cultural_cluster_id`: cluster cultural atribuído via IA (baseado em padrões de linguagem + localização).
- `interaction_history`: histórico de expressões e escolhas linguísticas.

📤 Saídas

- `localized_content`: conteúdo adaptado ao idioma e cultura.
- `tone_adjustment`: ajustes de tom (formalidade, proximidade, empatia).
- `idiomatic_variants`: expressões regionais aplicadas.
- `cultural_safety_flags`: alerta para conteúdos que possam ser ofensivos ou inadequados em uma região específica.

📊 Regras Técnicas

- **Fallback neutro**: se o idioma/região não estiver mapeado, adota-se padrão universal em inglês neutro.
- **Sensibilidade cultural**: dicionário de termos proibidos/inadequados por região.
- **Clusters regionais**: cada país/região é agrupado em clusters culturais para ajustar comunicação.
- **Treino contínuo**: IA aprende gírias e expressões novas a partir de interações reais.

🔧 Exemplo de Contrato de API

Requisição

`POST /i18n/adapt`

```
{
  "user_id": "uuid",
  "preferred_language": "pt-BR",
  "text_input": "Você gostaria de participar do evento?",
  "cultural_cluster_id": "latam_br"
}
```

Resposta

```
{
  "localized_content": "Bora pro evento?",
  "tone_adjustment": "informal",
  "idiomatic_variants": ["Bora", "partiu"],
  "cultural_safety_flags": []
}
```

🧪 Métricas de Observabilidade

- `language_distribution`: % de usuários por idioma ativo.
- `fallback_rate`: % de casos em que fallback neutro foi aplicado.
- `cultural_flag_triggered`: número de alertas por uso de termos inadequados.
- `adaptation_latency_ms`: tempo médio para processar adaptação linguística.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Captura idioma preferido + geo] --> B[Identifica cultural_cluster_id]
    B --> C[Aplica dicionário de termos + ajustes de tom]
    C --> D[Localiza conteúdo e expressões]
    D --> E[Exibe ao usuário + registra logs]
```

✨ Fechamento da Camada

A **Camada 17** garante que o FriendApp não seja apenas global, mas **glocal**: respeitando culturas locais, adaptando o idioma e transmitindo mensagens que façam sentido para cada pessoa.

Aqui, a IA Aurah Kosmos se torna **um tradutor sensível de mundos**, criando conexões que ultrapassam fronteiras linguísticas e culturais.

📦 CAMADA 18 — Telemetria & Observabilidade

🎯 Objetivo da Camada

Capturar, monitorar e analisar todos os **sinais técnicos e comportamentais** do FriendApp em tempo real, garantindo que a IA Aurah Kosmos opere com **baixa latência, alta disponibilidade e responsabilidade ética**.

Essa camada é o **painel nervoso central de monitoramento**, permitindo identificar falhas, abusos e desvios antes que impactem a experiência do usuário.

🔑 Entradas

- `request_logs`: logs de requisições REST/WebSocket.
- `latency_signals`: tempos de resposta em cada camada crítica.
- `error_events`: erros capturados no backend.
- `feedback_signals`: avaliações do usuário (“foi útil?”).
- `usage_metrics`: número de interações por feature.
- `ethical_counters`: contadores de ações proativas da IA (para evitar excesso de intervenção).

📤 Saídas

- `observability_dashboard`: dashboards em tempo real para DevOps e IA Ops.
- `alert_events`: alertas automáticos (ex.: p95 latência > 300ms).
- `ethical_reports`: relatórios de auditoria sobre decisões da IA.

- `health_score ∈ [0,1]`: indicador global de saúde do sistema.

📊 Regras Técnicas

- **Padrão OpenTelemetry** para coleta de sinais.
- **Dashboards com Prometheus + Grafana**, acessíveis apenas a DevOps/IA Ops.
- **Amostragem diferenciada**: 100% dos erros e 1% das interações normais para reduzir custo.
- **KPIs éticos monitorados**: nº de intervenções proativas da IA/dia, % de usuários que desativaram IA proativa.

🔧 Exemplo de Evento de Telemetria

```
{
  "timestamp": 1726479000,
  "user_id": "uuid",
  "session_id": "uuid_sess",
  "endpoint": "/feed",
  "latency_ms": 280,
  "status_code": 200,
  "cpu_usage_pct": 42.1,
  "memory_usage_mb": 310,
  "ai_proactive_actions": 1
}
```

🧪 Métricas de Observabilidade

- `p95_latency_ms`: latência em 95% das requisições.
- `error_rate`: % de requisições com erro.
- `uptime_pct`: disponibilidade do sistema.
- `ai_proactive_rate`: frequência de ações não solicitadas da IA.
- `user_feedback_score`: média de avaliações “foi útil?”.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Captura sinais de requests, erros e feedback] --> B[OpenTelemetry Co
llector]
    B --> C[Armazena no Prometheus/BigQuery]
    C --> D[Dashboards em Grafana]
    D --> E{KPIs fora do limite?}
    E -- Sim --> F[Dispara alerta para DevOps/IA Ops]
    E -- Não --> G[Operação normal]
```

✨ Fechamento da Camada

A **Camada 18** garante que o FriendApp não seja uma **caixa-preta**.

Cada ação técnica e cada decisão da IA é observada, registrada e auditada em tempo real.

Isso dá confiança de que o sistema é **confiável, ético e resiliente**, pronto para escalar sem perder qualidade ou segurança.

📦 CAMADA 19 — Logs Profundos (Privacidade)

🎯 Objetivo da Camada

Registrar e manter um **histórico profundo de interações** para aprendizado da IA Aurah Kosmos, mas sempre garantindo **anonimização, consentimento e proteção de dados pessoais (PII)**.

Essa camada é o coração da **memória técnica do sistema**, permitindo que a IA evolua com base em padrões coletivos sem comprometer a privacidade individual.

🔑 Entradas

- `interaction_events`: cliques, check-ins, mensagens, feedbacks.

- `system_signals`: latência, falhas, quedas de sessão.
- `feedback_signals`: respostas de utilidade dadas pelo usuário.
- `consent_flags`: status de consentimento ativo (opt-in/out).

📤 Saídas

- `deep_logs`: registros técnicos e comportamentais anonimizados.
- `user_opt_out_status`: indicador se os logs daquele usuário devem ser apagados.
- `semantic_reduced_logs`: logs antigos resumidos para economia de espaço.
- `compliance_reports`: relatórios para auditoria regulatória (LGPD/GDPR).

📊 Regras Técnicas

- **Anonimização total**: IDs reais são substituídos por `hash_user_id`.
- **TTL (time to live)**: logs detalhados expiram em 180 dias; após isso, passam por compressão semântica.
- **Opt-out imediato**: usuário pode desligar coleta de logs e pedir remoção instantânea.
- **Armazenamento segmentado**: PII em banco isolado; logs em data lake anônimo.
- **Acesso controlado**: apenas IA e equipe de auditoria têm acesso a relatórios.

🔧 Exemplo de Estrutura de Log

```
{
  "hash_user_id": "a98f123c",
  "session_id": "uuid_sess",
  "timestamp": 1726483000,
  "event": "checkin_event",
  "location_hash": "6gkz7",
  "emotional_state_score": 0.64,
  "feedback_signal": "useful"
```

```
}
```

🧪 Métricas de Observabilidade

- `log_volume_per_day`: volume médio de eventos registrados.
- `anonymization_success_rate`: % de logs sem PII.
- `opt_out_count`: número de usuários que desligaram coleta de logs.
- `compliance_audit_score`: conformidade com LGPD/GDPR.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Captura interações e sinais] --> B[Anonimização (hash_user_id)]
    B --> C[Persistência em data lake anônimo]
    C --> D[TTL: 180 dias]
    D --> E{Expirou?}
    E -- Sim --> F[Redução semântica + compressão]
    E -- Não --> G[Log ativo]
    F --> H[Disponível apenas para IA]
```

✨ Fechamento da Camada

A **Camada 19** garante que o FriendApp **aprenda sem invadir**.

A IA Aurah Kosmos ganha memória coletiva, mas nunca às custas da privacidade individual.

Cada usuário sabe que seus dados são tratados com **respeito, segurança e transparência**, mantendo o equilíbrio entre evolução tecnológica e confiança.

📦 CAMADA 20 — Consentimento & Ética Aplicada

🎯 Objetivo da Camada

Estabelecer os **princípios éticos e os mecanismos práticos de consentimento** que regem todo o ecossistema do FriendApp.

Essa camada garante que a IA Aurah Kosmos opere **sob governança clara, transparente e auditável**, colocando os direitos do usuário acima da conveniência técnica.

🔑 Entradas

- `user_id`: identificador único.
- `consent_flags`: flags de consentimento ativos (uso de dados, localização, IA proativa).
- `consent_version`: versão do termo de uso aceito.
- `feature_toggles`: recursos ativados/desativados pelo usuário.
- `audit_logs`: registros de mudanças em consentimentos.

📤 Saídas

- `consent_audit_trail`: trilha de auditoria de todas as decisões de consentimento.
- `ethical_reports`: relatórios automáticos sobre uso de dados e decisões críticas da IA.
- `policy_change_alerts`: notificações para usuários em caso de mudanças nas regras.
- `council_review_flags`: eventos críticos enviados ao Conselho Vibracional Humano.

📊 Regras Técnicas

- **Opt-in explícito**: nenhuma feature sensível (localização, IA proativa) pode ser ativada sem consentimento expresso.
- **Opt-out imediato**: ao desativar, recurso deve ser desligado em tempo real.
- **Change log público**: alterações em políticas devem ser versionadas e publicadas para auditoria.
- **Conselho Vibracional Humano**: toda decisão ética de alto impacto deve passar por revisão humana.

- **Trilha imutável**: cada ação de consentimento gera hash e é armazenada para fins de auditoria.

🔧 Exemplo de Contrato de API

Requisição (atualizar consentimento)

`POST /consent/update`

```
{
  "user_id": "uuid",
  "share_location": true,
  "ai_proactivity": false,
  "consent_version": "1.5"
}
```

Resposta

```
{
  "updated_flags": {
    "share_location": true,
    "ai_proactivity": false},
  "consent_audit_trail": [
    {"flag": "share_location", "status": true, "timestamp": 1726490000, "has
h": "abc123"}
  ]
}
```

🧪 Métricas de Observabilidade

- `opt_in_rate`: % de usuários que ativaram recursos sensíveis.
- `opt_out_rate`: % de usuários que desativaram recursos.
- `policy_change_ack_rate`: % de usuários que aceitaram nova versão de termos.
- `council_review_count`: nº de eventos encaminhados ao conselho humano.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Usuário ajusta consentimento] --> B[Atualiza consent_flags em tempo real]
    B --> C[Registra hash em audit trail]
    C --> D{Mudança crítica?}
    D -- Sim --> E[Encaminhar para Conselho Vibracional Humano]
    D -- Não --> F[Persistir ajuste]
```

✨ Fechamento da Camada

A **Camada 20** é o **código de ética vivo do FriendApp**.

Aqui, o usuário tem sempre a palavra final: pode aceitar, recusar ou revogar sua participação em qualquer fluxo.

A Aurah Kosmos não apenas segue essas regras — ela é **construída sobre elas**, garantindo que cada passo da IA seja auditável, humano e justo.

📦 CAMADA 21 — Orquestração de Recomendação (Planner)

🎯 Objetivo da Camada

Combinar dados de **perfil, contexto, estado emocional e compatibilidade** para criar um **plano de recomendações coerente**: pessoas, eventos, locais e experiências.

Essa camada funciona como o **motor de planejamento da IA Aurah Kosmos**, garantindo que as sugestões entregues ao usuário tenham **lógica, relevância e equilíbrio**.

🔑 Entradas

- `user_id`: identificador único.
- `compatibility_vector`: embedding social (Camada 06).
- `context_vector`: ambiente atual (Camada 07).

- `emotional_state_score`: estado atual (Camada 04).
- `match_score`: afinidade entre pares (Camada 06).
- `event_fit`: adequação a eventos/locais (Camada 10).
- `trust_score`: reputação social/técnica do usuário (Camada 35).

📤 Saídas

- `recommendation_plan`: lista priorizada de recomendações (pessoas, eventos, grupos, locais).
- `priority_score ∈ [0,1]`: score de prioridade de cada item no plano.
- `schedule_alignment`: ajustes de horário/contexto.
- `why_explanation`: justificativa clara para cada recomendação.

📊 Regras Técnicas

- **Diversidade forçada**: recomendações não podem ser todas do mesmo tipo (ex.: só pessoas, só eventos).
- **Prioridade ética**: recomendações de impacto social positivo têm boost.
- **Limite de volume**: máximo de 5 recomendações por ciclo para evitar overload.
- **Recalibração dinâmica**: se usuário rejeita várias sugestões, o planner ajusta pesos.

🔧 Fórmula Simplificada

```
priority_score = 0.4 * match_score
               + 0.3 * context_fit
               + 0.2 * expected_uplift
               + 0.1 * trust_score_norm
```

🔧 Exemplo de Contrato de API

Requisição

`POST /planner/recommend`

```
{
  "user_id": "uuid",
  "context_vector": [0.23, -0.11, 0.84],
  "match_score": 0.82,
  "event_fit": 0.77,
  "trust_score": 650
}
```

Resposta

```
{
  "recommendation_plan": [
    {
      "type": "person",
      "id": "uuid_peer",
      "priority_score": 0.81,
      "why_explanation": "Alta afinidade de perfil + proximidade contextual"
    },
    {
      "type": "event",
      "id": "event_123",
      "priority_score": 0.74,
      "why_explanation": "Evento de arte próximo + boost por interesse cultu
ral"
    }
  ]
}
```

🧪 Métricas de Observabilidade

- `acceptance_rate`: % de recomendações aceitas.
- `rejection_rate`: % de recomendações recusadas.
- `avg_priority_score`: média dos scores recomendados.
- `plan_diversity_index`: medida de diversidade (pessoas, eventos, locais).

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Captura embeddings e scores] --> B[Calcula priority_score]
    B --> C[Aplica regras de diversidade e ética]
    C --> D[Gera recommendation_plan]
    D --> E[Exibe recomendações com why_explanation]
    E --> F[Captura feedback do usuário]
    F --> G[Recalibra pesos e modelos]
```

✨ Fechamento da Camada

A **Camada 21** é o **cérebro estratégico da recomendação**: transforma múltiplos sinais em um **plano equilibrado de sugestões** que fazem sentido para o usuário.

Ela garante que o FriendApp não apenas recomende, mas **planeje experiências reais com propósito, diversidade e relevância**.

📦 CAMADA 22 — Feedback Pós-Interação

🎯 Objetivo da Camada

Capturar, registrar e processar o **feedback do usuário após interações reais** (chat, eventos, conexões, conteúdos do feed).

Essa camada fecha o **ciclo de aprendizado** da IA Aurah Kosmos, permitindo ajustar recomendações futuras e validar se a experiência foi **útil, segura e autêntica**.

🔑 Entradas

- `user_id`: identificador único.
- `interaction_type`: tipo de interação (chat, evento, feed, conexão).
- `interaction_id`: ID da interação específica.

- `feedback_signal`: avaliação explícita (útil/não útil, gostei/não gostei, estrelas).
- `text_feedback`: comentário opcional do usuário.
- `emotional_state_delta`: diferença do score emocional antes e depois da interação (Camada 04).

📤 Saídas

- `feedback_log`: registro completo da avaliação.
- `interaction_score ∈ [0,1]`: score calculado de qualidade da interação.
- `nps_social`: índice de satisfação social (baseado em feedback + delta emocional).
- `insights`: ajustes recomendados para IA com base no padrão do feedback.

📊 Regras Técnicas

- **Feedback contextual**: solicitado de forma leve após interações importantes (ex.: após um evento).
- **Não intrusivo**: usuário nunca deve sentir obrigação de responder.
- **Delta emocional**: comparado para medir se interação trouxe ganho ou retração.
- **Feedback explícito > implícito**: sempre priorizar o que o usuário disse sobre o que o modelo inferiu.

🔧 Fórmula de Score

```
interaction_score = 0.5 * feedback_signal_norm
                  + 0.3 * (1 - report_flag)
                  + 0.2 * emotional_state_delta_norm
```

- `feedback_signal_norm`: normalização da avaliação explícita.
- `report_flag`: binário (1 = interação denunciada, 0 = não).
- `emotional_state_delta_norm`: ganho/perda emocional normalizado.

🔧 Exemplo de Contrato de API

Requisição

`POST /feedback/interaction`

```
{
  "user_id": "uuid",
  "interaction_type": "evento",
  "interaction_id": "event_123",
  "feedback_signal": "useful",
  "text_feedback": "Gostei da vibe, me senti acolhido",
  "emotional_state_delta": 0.15
}
```

Resposta

```
{
  "interaction_score": 0.82,
  "nps_social": 9,
  "feedback_log": {
    "timestamp": 1726500000,
    "hash": "xyz123"
  }
}
```

🧪 Métricas de Observabilidade

- `feedback_response_rate`: % de usuários que responderam ao feedback.
- `avg_interaction_score`: média de qualidade das interações.
- `emotional_uplift_rate`: % de interações que geraram ganho emocional.
- `negative_feedback_rate`: proporção de feedbacks negativos.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Usuário conclui interação] --> B[Solicitação de feedback]
    B --> C[Captura feedback_signal + texto]
    C --> D[Calcula interaction_score]
    D --> E[Atualiza nps_social + insights]
    E --> F[Registra em feedback_log]
    F --> G[Ajusta recomendações futuras]
```

✨ Fechamento da Camada

A **Camada 22** é o **espelho do usuário dentro da IA**: é aqui que a Aurah Kosmos entende, diretamente pela voz e sensação do usuário, se suas sugestões realmente geraram valor.

Cada feedback recebido não é apenas um dado — é um **ato de co-criação da experiência**, tornando o FriendApp cada vez mais autêntico e humano.

📦 CAMADA 23 — Check-ins & Veracidade

🎯 Objetivo da Camada

Confirmar a **presença real do usuário em eventos e locais parceiros**, garantindo que as interações registradas no FriendApp correspondam a experiências autênticas.

Essa camada é essencial para dar **credibilidade às conexões**, alimentar corretamente o mapa de frequência e prevenir fraudes ou falsos check-ins.

🔑 Entradas

- `user_id`: identificador único.
- `event_id` ou `partner_location_id`: evento/local relacionado.
- `geo_hash`: localização aproximada do dispositivo.

- `proximity_signals`: sinais adicionais (Bluetooth Low Energy, QR code, Wi-Fi local).
- `timestamp`: horário do check-in.

📤 Saídas

- `checkin_verified`: flag binária (true/false).
- `checkin_log`: registro persistente da presença.
- `trust_score_delta`: ajuste na reputação do usuário (Camada 35).
- `engagement_signal`: insumo para gamificação (Camada 13) e eventos (Camada 10).

📊 Regras Técnicas

- **Geo + proximidade**: check-in só é válido se localização e pelo menos um sinal secundário (QR, BLE, Wi-Fi) confirmarem presença.
- **Falsificação barrada**: check-ins repetidos sem sinais de proximidade reduzem `trust_score`.
- **Participação real**: apenas check-ins confirmados contam para XP, eventos ou impacto social.
- **Privacidade preservada**: localização salva como `geo_hash`, nunca lat/long precisa.

🔧 Exemplo de Contrato de API

Requisição

`POST /events/checkin`

```
{
  "user_id": "uuid",
  "event_id": "event_123",
  "geo_hash": "6gkz7",
  "proximity_signals": {
    "qr_code": true,
    "ble_signal": -65
  },
  "timestamp": 1726512000
```

```
}
```

Resposta

```
{
  "checkin_verified": true,
  "checkin_log": {
    "event_id": "event_123",
    "geo_hash": "6gkz7",
    "timestamp": 1726512000
  },
  "trust_score_delta": +15,
  "engagement_signal": "evento_autenticado"
}
```

🧪 Métricas de Observabilidade

- `checkin_success_rate`: % de check-ins validados.
- `false_checkin_attempts`: tentativas de fraude barradas.
- `avg_checkin_latency_ms`: tempo médio de verificação.
- `trust_score_impact`: variação média no score de reputação após check-ins.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Usuário tenta check-in] --> B[Captura geo_hash + sinais proximidade]
    B --> C[Validação múltipla]
    C --> D{Check-in válido?}
    D -- Sim --> E[Grava log + ajusta trust_score]
    D -- Não --> F[Rejeita check-in + penaliza confiança]
    E --> G[Alimenta gamificação e eventos]
    F --> G
```

✨ Fechamento da Camada

A **Camada 23** garante que o FriendApp mantenha a **veracidade das experiências**.

Cada check-in validado reforça a confiança no ecossistema e impede que usuários ou parceiros tentem manipular métricas.

Aqui, presença é mais que um clique: é um **ato confirmado de pertencimento**.

📦 CAMADA 24 — Selos, Status & Alertas (UI)

🎯 Objetivo da Camada

Exibir ao usuário, de forma clara e vibracional, seus **selos conquistados**, **alertas ativos** e **status de reputação**.

Essa camada é a **interface visível da evolução** do usuário dentro do FriendApp, tornando palpável o que antes era apenas cálculo interno da IA Aurah Kosmos.

🔑 Entradas

- `user_id`: identificador único.
- `trust_score`: reputação do usuário (Camada 35).
- `xp_level`: nível de evolução (Camada 13).
- `badges`: selos conquistados (amizade autêntica, impacto social, presença).
- `alerts`: avisos de moderação, risco ou missão ativa.
- `verification_status`: status de verificação documental (DUC/DCO).

📤 Saídas

- `user_seals`: lista de selos ativos + status (ativo, suspenso, expirado).
- `user_status`: estado público (“Confiança Alta”, “Em observação”).
- `user_alerts`: notificações visíveis apenas para o próprio usuário.
- `ui_panels`: exibição de missões e recomendações ativas.

📊 Regras Técnicas

- **Selos ativos**: exibidos com animações sutis (pulso, brilho).
- **Selos suspensos**: exibidos em cinza com aura opaca.
- **Alertas privados**: somente o próprio usuário vê; nunca públicos.
- **Status público**: resumido em 3 níveis → “Alta Confiança”, “Confiança Média”, “Em Observação”.
- **Missão ativa**: exibida como banner sensorial leve.

🔧 Exemplo de Estrutura de Banco

Tabela `user_seals`

- `id_user`
- `seal_type` (amizade_autentica, impacto_social)
- `status` (ativo, suspenso, expirado)
- `date_earned`
- `expiration_date`

Tabela `user_alerts`

- `alert_type` (risco_chat, missão, confiança)
- `message`
- `timestamp`
- `dismissed` (true/false)

🔧 Exemplo de Contrato de API

Requisição

`GET /user/status?user_id=uuid`

Resposta

```
{
  "user_seals": [
    {"type": "amizade_autentica", "status": "ativo", "date_earned": "2025-08
-01"},
```

```
  {"type": "impacto_social", "status": "suspenso", "date_earned": "2025-0
7-10"}
 ],
 "user_status": {
  "trust_score": 820,
  "label": "Confiança Alta"
 },
 "user_alerts": [
  {"alert_type": "missao", "message": "Concluir missão de conexão", "time
stamp": 1726520000}
 ],
 "verification_status": "verificado"
}
```


🧪 Métricas de Observabilidade

- `badge_claim_rate`: % de usuários que exibem selos conquistados.
- `alert_resolution_time`: tempo médio para usuário resolver alerta.
- `status_distribution`: proporção de usuários em cada status público.
- `selo_suspension_rate`: frequência de suspensões de selos.

📋 Fluxo Lógico Simplificado

```
flowchart TD
  A[Captura trust_score + xp_level + verificações] --> B[Atualiza user_seal
s]
  B --> C[Define user_status público]
  A --> D[Gera user_alerts privados]
  C --> E[Renderiza painel de status no app]
  D --> E
```

✨ Fechamento da Camada

A **Camada 24** é a **vitrine da jornada vibracional**.

Aqui, conquistas são celebradas, riscos são comunicados de forma privada e a reputação se torna visível, tudo com estética clara e simbólica.

É a forma do FriendApp transformar **dados invisíveis em símbolos tangíveis de evolução e confiança**.

📦 CAMADA 25 — Trilhas (Itinerários)

🎯 Objetivo da Camada

Criar **sequências de experiências recomendadas** para o usuário, chamadas de **trilhas** ou **itinerários vibracionais**, que conectam pessoas, eventos e locais de forma coerente.

Essa camada transforma recomendações isoladas em **jornadas completas**, ajudando o usuário a viver experiências reais alinhadas com sua energia, contexto e objetivos pessoais.

🔑 Entradas

- `user_id`: identificador único.
- `recommendation_plan`: plano de recomendações (Camada 21).
- `context_vector`: ambiente atual (Camada 07).
- `compatibility_vector`: afinidade social (Camada 06).
- `event_fit`: adequação a eventos (Camada 10).
- `mission_status`: progresso em missões (Camada 59).

📤 Saídas

- `itinerary`: lista de recomendações encadeadas com horários e locais.
- `goal_label`: objetivo principal da trilha (ex.: socialização leve, expansão cultural, autocuidado).
- `priority_score`: score de adequação da trilha ao usuário.
- `why_explanation`: justificativa clara do itinerário.

📊 Regras Técnicas

- **Objetivo definido**: cada trilha tem um propósito (ex.: “expansão social”, “imersão cultural”).
- **Sequência coerente**: eventos devem ter lógica temporal e geográfica.
- **Diversidade**: trilha deve incluir pelo menos 2 tipos diferentes de experiências (ex.: evento + conexão).
- **Feedback em loop**: após a trilha, usuário avalia → IA ajusta itinerários futuros.

🔧 Fórmula Simplificada

```
priority_score = 0.4 * avg_event_fit
               + 0.3 * context_alignment
               + 0.2 * compatibility_boost
               + 0.1 * mission_relevance
```

🔧 Exemplo de Contrato de API

Requisição

`POST /trails/generate`

```
{
  "user_id": "uuid",
  "recommendation_plan": [
    {"type": "event", "id": "e123", "event_fit": 0.81},
    {"type": "person", "id": "p456", "match_score": 0.76}
  ],
  "mission_status": "expansao_social"
}
```

Resposta

```
{
  "itinerary": [
    {
      "step": 1,
```

```
      "type": "event",
      "id": "e123",
      "time": "19:00",
      "why_explanation": "Evento cultural próximo + alta afinidade com seu p
erfil"
    },
    {
      "step": 2,
      "type": "person",
      "id": "p456",
      "time": "21:00",
      "why_explanation": "Compatibilidade alta e interesse em socialização"
    }
  ],
  "goal_label": "expansao_social",
  "priority_score": 0.79
}
```

🧪 Métricas de Observabilidade

- `trail_completion_rate`: % de trilhas concluídas pelo usuário.
- `avg_priority_score`: média de adequação das trilhas geradas.
- `multi_type_ratio`: proporção de trilhas com diversidade de experiências.
- `feedback_trail_score`: avaliação média das trilhas concluídas.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Recebe recommendation_plan] --> B[Define objetivo da trilha]
    B --> C[Seleciona eventos + conexões coerentes]
    C --> D[Organiza em sequência temporal/espacial]
    D --> E[Calcula priority_score]
    E --> F[Entrega itinerary ao usuário]
    F --> G[Captura feedback pós-trilha]
```

```
    G --> H[Recalibra modelos futuros]
```

✨ Fechamento da Camada

A **Camada 25** é onde o FriendApp deixa de ser apenas um recomendador e se torna um **curador de jornadas**.

Aqui, a IA Aurah Kosmos não apenas sugere experiências: ela **organiza a vida do usuário em itinerários significativos**, que combinam propósito, diversidade e autenticidade.

📦 CAMADA 26 — Chat: Entradas e Saídas (Integrações)

🎯 Objetivo da Camada

Consolidar todas as **entradas que podem disparar uma conversa** e todas as **saídas que o chat gera para outros sistemas**.

Essa camada garante que o chat do FriendApp não seja isolado, mas sim um **hub de integração com feed, eventos, mapa, reputação e painel vibracional**.

🔑 Entradas

- **Feed Sensorial (Camada 08):** posts e interações podem gerar convites de conversa.
- **Eventos & Locais (Camada 10):** check-ins ativam grupos de chat coletivo.
- **Mapa de Frequência (Camada 12):** proximidade detectada gera sugestão de chat.
- **Conexões Autênticas (Camada 09):** match validado abre canal de chat direto.
- **Notificações Vibracionais (Camada 31):** lembretes e alertas podem incluir links para chat.
- **Modo Viagem + Bora (Camada 32):** grupos temporários criam chats de itinerário.

📤 Saídas

- **Painel Vibracional (Premium, Camada 24):** registros de chats autênticos alimentam conexões vibracionais.
- **Sistema de Reputação (Camada 35):** interações positivas ou negativas impactam `trust_score`.
- **Gamificação (Camada 13):** mensagens de qualidade geram XP ou selos.
- **IA de Moderação (Camada 15):** mensagens de risco são encaminhadas para análise imediata.
- **Feedback Pós-Interação (Camada 22):** chat gera métricas de utilidade e autenticidade.
- **Logs Profundos (Camada 19):** dados anonimizados para aprendizado coletivo.

📊 Regras Técnicas

- **Ativação mínima:** chat só abre após intenção validada dos dois lados (consentimento duplo).
- **Integração em tempo real:** via WebSocket, garantindo baixa latência.
- **Logs segmentados:** mensagens privadas → apenas entre usuários + moderação; metadados → enviados para IA e métricas.
- **Explicabilidade:** usuário sabe por que o chat foi sugerido (“Vocês participaram do mesmo evento ontem”).

🔧 Exemplo de Contrato de API

Requisição (abrir chat)

`POST /chat/open`

```
{
  "user_a": "uuid_A",
  "user_b": "uuid_B",
  "reason": "evento_compartilhado"
}
```

Resposta

```
{
  "chat_id": "chat_123",
  "status": "active",
  "why_explanation": "Vocês participaram do mesmo evento de música em São Paulo"
}
```

🧪 Métricas de Observabilidade

- `chat_open_rate`: % de chats sugeridos que foram aceitos.
- `avg_message_length`: média de palavras por mensagem (indicador de profundidade).
- `moderation_trigger_rate`: % de mensagens enviadas para moderação.
- `conversion_to_connection`: % de chats que viraram conexões autênticas.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Feed, Eventos, Mapa, Conexões] --> B[Trigger de chat]
    B --> C[Validação de consentimento duplo]
    C --> D[Abrir canal de chat]
    D --> E[Mensagens em tempo real via WS]
    E --> F[Integrações: reputação, gamificação, logs]
```

✨ Fechamento da Camada

A **Camada 26** é o **nó integrador do FriendApp**.

Cada conversa aberta não é apenas um chat, mas um **elo que conecta feed, eventos, mapa e reputação**.

Aqui, o chat deixa de ser um recurso isolado e se torna um **mecanismo vivo de integração social e técnica**.

📦 CAMADA 27 — Doações & Impacto Social (Integrações)

🎯 Objetivo da Camada

Gerenciar as **doações feitas dentro do FriendApp** e transformá-las em **impacto social mensurável e transparente**.

Essa camada conecta a jornada individual com o coletivo: cada contribuição vira um registro público de impacto, fortalecendo a missão do app de unir tecnologia e transformação social.

🔑 Entradas

- `user_id`: identificador único.
- `donation_value`: valor doado em reais ou FriendCoins.
- `donation_target`: causa, ONG ou projeto parceiro.
- `payment_metadata`: dados do pagamento (token seguro, não PII).
- `impact_event`: registros de entrega de impacto (ex.: cesta básica entregue, hora de voluntariado).

📤 Saídas

- `donation_log`: registro da doação (anonimizado ou público, conforme escolha do usuário).
- `impact_badge`: selo simbólico de impacto social conquistado.
- `transparency_report`: relatório consolidado de impacto coletivo.
- `trust_score_delta`: aumento na reputação do usuário (Camada 35).

📊 Regras Técnicas

- **Pagamentos seguros**: integração com gateways (Pix, Stripe, PayPal).
- **Conversão FriendCoins → impacto**: moedas podem ser usadas em doações internas.

- **Selo de impacto**: cada contribuição gera reconhecimento simbólico, não competitivo.
- **Relatórios públicos**: todo impacto é consolidado em painel acessível a todos os usuários.
- **Opt-in de anonimato**: usuário pode escolher aparecer ou não nos relatórios públicos.

🔧 Exemplo de Contrato de API

Requisição

`POST /donations/create`

```
{
  "user_id": "uuid",
  "donation_value": 50,
  "donation_target": "ong_educacao",
  "payment_metadata": {"payment_token": "tok_12345"}
}
```

Resposta

```
{
  "donation_log": {
    "amount": 50,
    "currency": "BRL",
    "target": "ong_educacao",
    "timestamp": 1726530000,
    "hash": "abc123xyz"
  },
  "impact_badge": "Educador Social",
  "trust_score_delta": +20
}
```

🧪 Métricas de Observabilidade

- `donation_volume_total`: total arrecadado em reais/FriendCoins.

- `impact_conversion_rate`: % de doações convertidas em entregas sociais.
- `anon_donation_rate`: proporção de doações feitas anonimamente.
- `trust_score_impact_avg`: variação média na reputação após doações.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Usuário realiza doação] --> B[Gateway de pagamento seguro]
    B --> C[Registra donation_log]
    C --> D[Converte em impacto social concreto]
    D --> E[Gera impact_badge + trust_score_delta]
    E --> F[Exibe no painel de impacto coletivo]
```

✨ Fechamento da Camada

A **Camada 27** conecta **propósito individual ao impacto coletivo**.

Aqui, cada ato de generosidade se transforma em dado transparente e ação concreta, reforçando que o FriendApp não é apenas um app de conexões — é também uma **plataforma de transformação social autêntica**.

📦 CAMADA 28 — Parceiros & Painel B2B

🎯 Objetivo da Camada

Gerenciar a **integração de locais parceiros** (cafés, espaços de coworking, casas de cultura, academias, etc.) e oferecer a eles um **Painel B2B** para acompanhar engajamento, métricas e resultados de suas ativações no FriendApp.

Essa camada conecta a economia do app com o mundo físico, criando **valor para empresas e experiências para usuários**.

🔑 Entradas

- `partner_id`: identificador único do parceiro.

- `location_data`: dados do local (geo_hash, categoria, horários, capacidade).
- `event_metadata`: eventos criados pelo parceiro.
- `user_checkins`: presença de usuários confirmada no local (Camada 23).
- `boost_campaigns`: impulsionamentos pagos (via FriendCoins ou Premium).
- `user_feedback`: avaliações e reputação do local.

📤 Saídas

- `partner_dashboard`: painel consolidado com métricas de engajamento.
- `location_visibility_score`: índice de visibilidade do parceiro no app.
- `event_performance_report`: relatório de performance de eventos.
- `billing_logs`: registro de transações e créditos.

📊 Regras Técnicas

- **Check-ins como prova de valor**: visibilidade do parceiro só conta se houver presença real confirmada.
- **Boost controlado**: impulsionamento aumenta visibilidade, mas nunca substitui afinidade vibracional.
- **Métricas transparentes**: parceiro vê quantos usuários passaram, interagiram e avaliaram.
- **Reputação dinâmica**: locais com feedbacks negativos são automaticamente rebaixados no ranking.

🔧 Exemplo de Contrato de API

Requisição (criar evento parceiro)

`POST /partner/events/create`

```
{
  "partner_id": "partner_123",
  "event_name": "Workshop de Criatividade",
  "geo_hash": "6gkz7",
  "tags": ["arte", "inovacao"],
  "time": "2025-09-20T19:00:00Z"
```

```
}
```

Resposta

```
{
  "event_id": "event_456",
  "status": "ativo",
  "visibility_score": 0.73
}
```

Requisição (consultar dashboard)

`GET /partner/dashboard?partner_id=partner_123`

Resposta

```
{
  "location_visibility_score": 0.81,
  "event_performance_report": [
    {"event_id": "event_456", "checkins": 120, "avg_feedback": 4.5}
  ],
  "billing_logs": [
    {"amount": 200, "currency": "FriendCoins", "reason": "impulsionamento_
evento"}
  ]
}
```

🧪 Métricas de Observabilidade

- `partner_retention_rate`: % de parceiros que permanecem ativos após 3 meses.
- `avg_visibility_score`: visibilidade média dos locais parceiros.
- `event_conversion_rate`: % de usuários sugeridos que realmente compareceram.
- `boost_usage_rate`: frequência de campanhas de impulsionamento.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Parceiro cadastra local/evento] --> B[IA calcula visibilidade inicial]
    B --> C[Usuários recebem recomendações no app]
    C --> D[Check-ins confirmam participação]
    D --> E[Dashboard atualiza métricas em tempo real]
    E --> F[Parceiro pode impulsionar campanhas]
    F --> G[Billing registra FriendCoins/real]
```

✨ Fechamento da Camada

A **Camada 28** é o ponto de encontro entre **o FriendApp e os parceiros comerciais**.

Ela garante que cada local parceiro tenha **ferramentas claras e métricas reais** para medir impacto, enquanto mantém a integridade do ecossistema: só aparece quem realmente entrega valor aos usuários.

📦 CAMADA 29 — Denúncias & Governança de Comunidade

🎯 Objetivo da Camada

Oferecer ao usuário **um canal simples e transparente para denúncias** e estabelecer os mecanismos de **governança comunitária** que asseguram que o FriendApp seja um espaço **ético, seguro e saudável**.

Essa camada é o **pilar democrático do ecossistema**, garantindo que a voz da comunidade seja ouvida e que decisões de moderação não sejam arbitrárias.

🔑 Entradas

- `user_id`: identificador único.
- `content_id`: identificador do conteúdo denunciado (post, chat, evento).
- `report_reason`: motivo da denúncia (assédio, fake, discurso de ódio, spam, fraude).

- `report_comment`: comentário opcional do usuário.
- `report_evidence`: captura de tela ou metadados opcionais.
- `report_timestamp`: horário da denúncia.

📤 Saídas

- `report_log`: registro da denúncia com hash para auditoria.
- `moderation_queue`: item encaminhado para IA de moderação (Camada 15).
- `council_review_flags`: casos críticos escalonados para o Conselho Vibracional Humano (Camada 20).
- `status_update`: retorno para o usuário sobre andamento (em análise, resolvido).

📊 Regras Técnicas

- **Fluxo ágil**: denúncia deve poder ser feita em até 3 cliques.
- **Feedback transparente**: usuário recebe atualizações sobre status da denúncia.
- **Proteção contra abuso**: denúncias falsas ou maliciosas reduzem `trust_score`.
- **Escalonamento humano**: casos de autoagressão, assédio grave ou fraude sistêmica vão para revisão manual.
- **Auditoria pública**: relatório agregado publicado periodicamente (sem expor dados individuais).

🔧 Exemplo de Contrato de API

Requisição (criar denúncia)

`POST /reports/create`

```
{
  "user_id": "uuid",
  "content_id": "post_123",
  "report_reason": "assédio",
  "report_comment": "Comentário ofensivo recebido no chat",
  "report_timestamp": 1726540000
```

```
}
```

Resposta

```
{
  "report_log": {
    "hash": "abc123xyz",
    "status": "em_analise"
  },
  "moderation_queue": true,
  "council_review_flags": false}
```

🧪 Métricas de Observabilidade

- `report_volume_daily`: nº de denúncias criadas por dia.
- `avg_resolution_time`: tempo médio para resolver uma denúncia.
- `false_report_rate`: % de denúncias consideradas maliciosas.
- `council_review_count`: nº de casos enviados para revisão humana.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Usuário envia denúncia] --> B[Registro em report_log com hash]
    B --> C[Encaminha p/ IA de moderação]
    C --> D{Gravidade alta?}
    D -- Sim --> E[Escalonar p/ Conselho Humano]
    D -- Não --> F[IA decide ação automática]
    E --> G[Status atualizado p/ usuário]
    F --> G
```

✨ Fechamento da Camada

A **Camada 29** é a **voz da comunidade transformada em sistema**.

Aqui, cada denúncia vira um processo auditável, transparente e justo, equilibrando a ação automática da IA com a revisão ética de humanos.

O resultado é um FriendApp onde **ninguém está invisível e a comunidade se protege coletivamente**.

📦 CAMADA 30 — Check-in Vibracional (Definição Técnica)

🎯 Objetivo da Camada

Registrar **check-ins de estado vibracional** do usuário, combinando auto-relatos leves com sinais passivos do app.

Essa camada não mede apenas **presença física** (Camada 23), mas o **estado interno** do usuário em um momento específico, criando dados que alimentam o **Mapa de Frequência (Camada 12)** e a **IA Aurah Kosmos**.

🔑 Entradas

- `user_id`: identificador único.
- `self_report`: auto-relato leve do usuário (“me sinto bem”, “cansado”, escala 1–5).
- `ui_signals`: tempo de tela, cliques, hesitação (Camada 05).
- `chat_emotions`: análise de sentimentos em mensagens recentes (Camada 04).
- `context_vector`: ambiente vivo (Camada 07).
- `checkin_timestamp`: horário do registro.

📤 Saídas

- `vibrational_checkin`: registro consolidado do estado do usuário.
- `emotional_state_score`: score atualizado (recalculado).
- `confidence_score ∈ [0,1]`: confiança do modelo no check-in.
- `checkin_log`: persistência para histórico e Mapa de Frequência.

📊 Regras Técnicas

- **Peso reduzido**: auto-relatos valem, mas não dominam o cálculo → combinados com sinais passivos.
- **Voluntariedade**: usuário pode pular o check-in sem penalidade.
- **Privacidade**: todos os registros são anonimizados ao alimentar agregados coletivos.
- **Periodicidade sugerida**: no máximo 2 check-ins voluntários por dia.

🔧 Fórmula de Recalibração

```
emotional_state_score_final =
   0.5 * self_report_norm
 + 0.3 * nlp_emotion
 + 0.2 * ui_signal_variation
```

🔧 Exemplo de Contrato de API

Requisição

`POST /checkin/vibrational`

```
{
  "user_id": "uuid",
  "self_report": 3,
  "ui_signals": {"hesitation_ms": 2200, "actions_per_min": 4},
  "chat_emotions": {"valence": -0.2, "arousal": 0.5},
  "checkin_timestamp": 1726548000
}
```

Resposta

```
{
  "vibrational_checkin": true,
  "emotional_state_score": 0.58,
  "confidence_score": 0.79,
```

```
  "checkin_log": {
    "timestamp": 1726548000,
    "location": "geo_hash:6gkz7"
  }
}
```

🧪 Métricas de Observabilidade

- `checkin_participation_rate`: % de usuários que realizam check-in vibracional.
- `avg_self_report_score`: média dos auto-relatos.
- `correlation_self_report_vs_signals`: correlação entre relatos e sinais passivos.
- `false_positive_rate`: check-ins incoerentes com sinais detectados.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Usuário envia self_report] --> B[Captura sinais passivos (UI + chat)]
    B --> C[Normalização dos dados]
    C --> D[Recalcula emotional_state_score]
    D --> E[Grava checkin_log]
    E --> F[Atualiza Mapa de Frequência]
```

✨ Fechamento da Camada

A **Camada 30** é o momento em que o usuário pode **parar e se observar** dentro do FriendApp.

Mais do que dados, o check-in vibracional é um convite à autoconsciência, transformado em insumo técnico para IA e para o coletivo.

Aqui, cada registro individual contribui para um **ecossistema mais consciente e humano**.

📦 CAMADA 31 — Notificações & Gatilhos (Respeitosos)

🎯 Objetivo da Camada

Gerenciar todas as **notificações enviadas ao usuário** de forma **respeitosa, transparente e configurável**, evitando sobrecarga ou sensação de vigilância.

Essa camada garante que a comunicação do FriendApp seja **oportuna, útil e nunca invasiva**.

🔑 Entradas

- `user_id`: identificador único.
- `notification_type`: tipo de notificação (evento, conexão, missão, alerta).
- `context_vector`: ambiente atual (Camada 07).
- `collapse_risk`: risco de colapso (Camada 67).
- `user_prefs`: preferências de notificação definidas pelo usuário.
- `quiet_hours`: janelas de silêncio configuradas pelo usuário.

📤 Saídas

- `notification_payload`: mensagem a ser entregue ao dispositivo.
- `delivery_channel`: canal utilizado (push, in-app, e-mail).
- `delivery_status`: confirmação de entrega.
- `audit_log`: registro da notificação enviada (com hash).

📊 Regras Técnicas

- **Respeito absoluto**: no máximo 3 notificações/dia por usuário, salvo alertas críticos.
- **Quiet Hours**: notificações desativadas em horários configurados (ex.: 22h–8h).
- **Opt-in granular**: usuário escolhe quais tipos de notificações deseja receber.

- **Gatilhos éticos**: missões regenerativas só são notificadas se risco ≥ threshold E houver opt-in.
- **Transparência**: cada notificação traz a explicação “por que você está vendo isso”.

🔧 Exemplo de Contrato de API

Requisição (gerar notificação)

`POST /notifications/send`

```
{
  "user_id": "uuid",
  "notification_type": "evento",
  "context_vector": [0.23, -0.11, 0.84],
  "message": "Novo evento de música perto de você amanhã",
  "delivery_channel": "push"
}
```

Resposta

```
{
  "delivery_status": "success",
  "audit_log": {
    "hash": "notif_abc123",
    "timestamp": 1726552000
  }
}
```

🧪 Métricas de Observabilidade

- `notification_volume_daily`: nº de notificações enviadas por dia.
- `open_rate`: % de notificações abertas.
- `quiet_hours_respected`: proporção de notificações bloqueadas por horário de silêncio.
- `opt_out_rate`: % de usuários que desativaram notificações.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Evento ou conexão detectada] --> B[Verifica prefs e quiet hours]
    B --> C{Permissão ativa?}
    C -- Não --> D[Descarta notificação]
    C -- Sim --> E[Gera notification_payload]
    E --> F[Envia via push/in-app/email]
    F --> G[Registra audit_log]
```

✨ Fechamento da Camada

A **Camada 31** assegura que o FriendApp fale com o usuário **com gentileza e clareza**.

Notificações não são gatilhos de ansiedade, mas lembretes respeitosos que reforçam a experiência de pertencimento.

Aqui, cada mensagem entregue é **um convite, nunca uma imposição**.

📦 CAMADA 32 — Modo Viagem + Bora (Conexões Reais)

🎯 Objetivo da Camada

Ativar o **modo de conexões presenciais** para usuários que estão em **viagem** ou querem criar encontros espontâneos com o recurso **Bora**.

Essa camada conecta **intenção, localização e afinidade social**, permitindo que pessoas se encontrem de forma autêntica em novas cidades ou contextos.

🔑 Entradas

- `user_id`: identificador do viajante.
- `trip_destination`: cidade/região de destino.
- `trip_dates`: intervalo da viagem.

- `context_vector`: ambiente vivo do usuário (Camada 07).
- `compatibility_vector`: afinidade social (Camada 06).
- `bora_group_requests`: pedidos de criação/participação em grupos Bora.
- `event_metadata`: eventos disponíveis no destino.

📤 Saídas

- `travel_profile`: perfil temporário de viagem (ativado apenas no destino).
- `bora_groups`: grupos criados para encontros reais.
- `connection_suggestions`: pares ou grupos compatíveis no destino.
- `pre_trip_notifications`: notificações antecipadas sobre conexões/eventos.
- `travel_logs`: registro de interações durante a viagem.

📊 Regras Técnicas

- **Ativação consciente**: modo viagem só é ativado se usuário declarar destino + datas.
- **Radar transparente**: viajante aparece no mapa de destino com ícone translúcido (cresce conforme a data da viagem se aproxima).
- **Consentimento duplo**: para entrar em grupo Bora, precisa aceitar o convite.
- **Integração com check-ins**: presença em Bora ou evento só é validada após check-in real (Camada 23).
- **Privacidade garantida**: localização precisa nunca é exibida, apenas região aproximada.

🔧 Exemplo de Contrato de API

Requisição (ativar viagem)

`POST /travel/activate`

```
{
  "user_id": "uuid",
  "trip_destination": "Lisboa",
  "trip_dates": {"start": "2025-09-20", "end": "2025-09-25"}
```

```
}
```

Resposta

```
{
  "travel_profile": {
    "destination": "Lisboa",
    "status": "ativo",
    "visible_from": "2025-09-10"
  },
  "pre_trip_notifications": [
    "Você e Ana estarão em Lisboa na mesma semana"
  ]
}
```

Requisição (criar grupo Bora)

POST /bora/create

```
{
  "user_id": "uuid",
  "title": "Bora pro café astral?",
  "tags": ["cultura", "networking"],
  "location": "geo_hash:6gkz7"
}
```

Resposta

```
{
  "bora_group_id": "bora_123",
  "status": "aberto",
  "participants": ["uuid", "uuid2"]
}
```

🧪 Métricas de Observabilidade

- `travel_mode_activation_rate`: % de usuários que ativaram modo viagem.
- `bora_group_creation_rate`: nº médio de grupos Bora criados por semana.
- `travel_connection_success_rate`: % de conexões realizadas em viagem.
- `drop_off_post_trip`: taxa de usuários que se desconectam após viagem.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Usuário ativa modo viagem] --> B[Cria travel_profile]
    B --> C[Exibe radar translúcido no destino]
    C --> D[Pré-sugestões de conexões e eventos]
    D --> E[Usuário cria ou entra em grupo Bora]
    E --> F[Check-ins confirmam encontros reais]
    F --> G[Registra travel_logs + ajusta reputação]
```

✨ Fechamento da Camada

A **Camada 32** é o motor do FriendApp para **encontros presenciais e viagens**.

Ela transforma intenção em realidade: seja em um **café local** ou em uma **cidade desconhecida**, o app conecta pessoas de forma planejada ou espontânea, sempre com segurança e consentimento.

📦 CAMADA 33 — UI/UX Sensorial (Adaptação Mensurável)

🎯 Objetivo da Camada

Adaptar a **interface visual e sensorial do FriendApp** de acordo com o estado emocional do usuário (`emotional_state_score`), mas de forma **mensurável, respeitosa e opt-in**.

Essa camada transforma a UI em um **espelho dinâmico da jornada**, reforçando estados positivos e reduzindo estímulos quando detectado risco de colapso.

🔑 Entradas

- `user_id`: identificador único.
- `emotional_state_score`: estado emocional (Camada 04).
- `attention_hesitation_ms`: tempo de hesitação do usuário (Camada 05).
- `context_vector`: ambiente atual (Camada 07).
- `feedback_signal`: ajustes manuais do usuário (“prefiro mais claro”, “prefiro escuro”).

📤 Saídas

- `ui_theme`: tema adaptado (claro, escuro, minimalista).
- `ui_intensity`: nível de estímulos visuais (alto, médio, baixo).
- `animation_state`: intensidade de animações (suave, neutro, reduzido).
- `why_explanation`: mensagem clara (“Interface suavizada porque detectamos cansaço”).

📊 Regras Técnicas

- **Sempre opt-in**: adaptações só ocorrem se usuário aceitar nas preferências.
- **Sutileza**: mudanças leves → não alterar radicalmente a UI.
- **Explicabilidade**: toda mudança vem acompanhada de explicação.
- **Fallback neutro**: se não houver dados ou usuário desativar, usar tema padrão.

🔧 Fórmula Simplificada

```
ui_intensity =
    if emotional_state_score < 0.3 then "baixo"
    else if hesitation_ms > 3000 then "baixo"
    else "médio"
```

🔧 Exemplo de Contrato de API

Requisição (atualizar UI)

`POST /ui/adapt`

```
{
  "user_id": "uuid",
  "emotional_state_score": 0.25,
  "hesitation_ms": 3200,
  "context_vector": [0.23, -0.11, 0.84]
}
```

Resposta

```
{
  "ui_theme": "escuro",
  "ui_intensity": "baixo",
  "animation_state": "reduzido",
  "why_explanation": "Interface suavizada por sinais de cansaço"
}
```

🧪 Métricas de Observabilidade

- `ui_adaptation_rate`: % de sessões com UI adaptada.
- `user_override_rate`: % de vezes que usuário reverteu a UI adaptada.
- `avg_hesitation_delta`: variação de hesitação antes e depois da adaptação.
- `feedback_acceptance`: % de usuários que consideraram útil a mudança.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Captura emotional_state + hesitação] --> B[Verifica opt-in do usuário]
    B -- Não --> C[Interface padrão]
    B -- Sim --> D[Calcula ajustes sutis]
    D --> E[Aplica tema/intensidade/animações]
    E --> F[Exibe why_explanation]
```

```
    F --> G[Captura feedback do usuário]
```

✨ Fechamento da Camada

A **Camada 33** transforma a UI do FriendApp em um **organismo vivo**: responsivo, ético e transparente.

Não se trata de manipular, mas de **suavizar ou energizar a experiência quando necessário**, sempre sob controle do usuário.

Aqui, a estética se torna também um **instrumento de cuidado**.

📦 CAMADA 35 — Reputação Técnica & Social

🎯 Objetivo da Camada

Construir e atualizar continuamente o **índice de reputação do usuário (`trust_score`)**, refletindo tanto a **confiabilidade técnica** (verificação, ausência de fraudes, cumprimento de regras) quanto a **reputação social** (qualidade das interações, feedbacks positivos, impacto coletivo).

Essa camada é o **pilar de confiança do FriendApp**, usada como base para desbloquear recursos, ajustar recomendações e moderar comportamentos.

🔑 Entradas

- `user_id`: identificador único.
- `duc_dco_status`: verificação documental concluída ou não.
- `checkin_verified`: presenças validadas em eventos (Camada 23).
- `report_flags`: denúncias contra o usuário (Camada 29).
- `feedback_logs`: avaliações recebidas de outros usuários (Camada 22).
- `donation_logs`: registros de impacto social (Camada 27).
- `xp_level`: evolução no jogo social (Camada 13).

📤 Saídas

- `trust_score ∈ [0,1000]`: índice atualizado de reputação.
- `trust_label`: categorização (“Alta Confiança”, “Média”, “Em Observação”).
- `trust_history`: histórico temporal de evolução da reputação.
- `suspension_flags`: alertas em casos de quedas críticas.

📊 Regras Técnicas

- **Pontuação inicial**: todo usuário começa com `trust_score = 500`.
- **Reforços positivos**: verificações, check-ins e feedbacks de qualidade aumentam score.
- **Penalizações**: denúncias confirmadas, tentativas de fraude ou spam reduzem score.
- **Limites públicos**:
  - `>800` → Confiança Alta.
  - `500–800` → Confiança Média.
  - `<500` → Em Observação.
- **Escalonamento**: usuários <300 podem ser suspensos até revisão.

🔧 Fórmula Simplificada

```
trust_score =
  400 * verifications
+ 300 * no_abuse_factor
+ 200 * engagement_quality
+ 100 * tenure_norm
```

- `verifications`: peso para DUC/DCO e check-ins validados.
- `no_abuse_factor`: reduz score em caso de denúncias confirmadas.
- `engagement_quality`: média ponderada dos feedbacks recebidos.
- `tenure_norm`: tempo de permanência ativo no app, normalizado.

🔧 Exemplo de Contrato de API

Requisição

`GET /reputation/status?user_id=uuid`

Resposta

```
{
  "trust_score": 820,
  "trust_label": "Alta Confiança",
  "trust_history": [
    {"timestamp": 1726560000, "score": 790},
    {"timestamp": 1726565000, "score": 820}
  ],
  "suspension_flags": []
}
```

🧪 Métricas de Observabilidade

- `avg_trust_score`: média global do índice de reputação.
- `distribution_curve`: distribuição de usuários por faixa de confiança.
- `false_penalty_rate`: penalizações revertidas após revisão humana.
- `suspension_rate`: % de usuários suspensos por score baixo.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Captura verificações, check-ins, feedbacks] --> B[Normaliza fatores]
    B --> C[Calcula trust_score]
    C --> D[Define trust_label]
    D --> E{trust_score < 300?}
    E -- Sim --> F[Dispara suspensão_flag + revisão]
    E -- Não --> G[Atualiza trust_history]
```

✨ Fechamento da Camada

A **Camada 35** é o **selo de confiança do ecossistema FriendApp**.

Ela garante que interações sejam respaldadas por um histórico claro e justo, premiando quem contribui positivamente e restringindo quem ameaça a comunidade.

Aqui, confiança deixa de ser abstrata e se torna **um índice vivo, auditável e justo**.

📦 CAMADA 36 — Anti-Manipulação & Spam

🎯 Objetivo da Camada

Detectar e bloquear **comportamentos maliciosos** dentro do FriendApp, como spam, bots, tentativas de manipulação de reputação ou criação artificial de conexões.

Essa camada garante que a **autenticidade** das interações não seja corrompida por uso indevido ou exploração do sistema.

🔑 Entradas

- `user_id`: identificador único.
- `interaction_events`: mensagens, posts, convites, check-ins.
- `frequency_signals`: volume e frequência de interações.
- `pattern_logs`: padrões suspeitos (repetição, horários anormais, cliques em massa).
- `trust_score`: reputação do usuário (Camada 35).
- `report_flags`: denúncias de spam recebidas (Camada 29).

📤 Saídas

- `spam_flag`: indicador binário (true/false).
- `risk_label`: nível de risco (baixo, médio, alto).
- `mitigation_actions`: medidas aplicadas (alerta, bloqueio temporário, suspensão).

- `audit_logs`: registro de detecção e resposta.

📊 Regras Técnicas

- **Rate limiting**: limite de mensagens, convites e posts por minuto.
- **Detecção de bots**: análise de padrões temporais (ex.: 200 mensagens em 5 minutos).
- **Farm de XP**: sistema anti-gamificação (Camada 13) cruza eventos para evitar pontos fraudulentos.
- **Denúncias aceleradoras**: múltiplas denúncias reduzem threshold para ação.
- **Registro imutável**: todas as detecções ficam salvas para auditoria futura.

🔧 Fórmula Simplificada

```
spam_risk_score =
  0.4 * frequency_anomaly
+ 0.3 * pattern_similarity
+ 0.2 * trust_score_penalty
+ 0.1 * report_density
```

- Se `spam_risk_score ≥ 0.8` → bloqueio automático.
- Se `0.5 ≤ spam_risk_score < 0.8` → alerta + revisão humana.
- Se `<0.5` → monitoramento passivo.

🔧 Exemplo de Contrato de API

Requisição

`POST /spam/detect`

```
{
  "user_id": "uuid",
  "interaction_events": 180,
  "frequency_signals": {"messages_per_min": 60},
  "pattern_logs": {"repeated_text": true},
```

```
  "trust_score": 420,
  "reports": 5
}
```

Resposta

```
{
  "spam_flag": true,
  "risk_label": "alto",
  "mitigation_actions": ["bloqueio_temporario", "escalonamento_humano"],
  "audit_logs": [
    {"timestamp": 1726570000, "reason": "mensagens em massa", "hash":
"spam123"}
  ]
}
```

🧪 Métricas de Observabilidade

- `spam_detection_rate`: % de spams detectados automaticamente.
- `false_positive_rate`: % de usuários marcados erroneamente como spam.
- `avg_response_time`: tempo médio entre detecção e ação.
- `suspension_count`: nº de contas suspensas por spam/manipulação.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Captura interações do usuário] --> B[Analisa frequência + padrões]
    B --> C[Calcula spam_risk_score]
    C --> D{Score ≥ threshold?}
    D -- Alto --> E[Spam_flag = true + bloqueio automático]
    D -- Médio --> F[Alerta + revisão humana]
    D -- Baixo --> G[Monitoramento passivo]
    E --> H[Registro em audit_logs]
    F --> H
```

```
G ⟶ H
```

✨ Fechamento da Camada

A **Camada 36** é o **anticorpo do FriendApp**.

Ela protege o ecossistema contra manipulação e abuso, garantindo que os espaços sociais sejam ocupados apenas por interações genuínas.

Aqui, cada conexão continua sendo **autêntica, justa e confiável**.

📦 CAMADA 37 — Cold Start (Usuário & Conteúdo)

🎯 Objetivo da Camada

Resolver o problema de **início de jornada** no FriendApp: quando há **poucos dados** sobre um usuário novo ou sobre um conteúdo recém-publicado.

Essa camada garante que mesmo em cenários de baixa informação inicial, a IA Aurah Kosmos consiga oferecer **recomendações relevantes, seguras e engajantes**.

🔑 Entradas

- `new_user_profile`: dados básicos de cadastro (Camada 02).
- `personality_vector`: quando incompleto, valores parciais do teste (Camada 03).
- `interaction_history`: vazio ou mínimo (apenas primeiros cliques).
- `new_content_metadata`: título, tags, tipo de mídia, autor.
- `population_baseline`: dados coletivos de comportamento médio.

📤 Saídas

- `bootstrap_recommendations`: primeiras recomendações para o usuário.
- `bootstrap_visibility`: nível de exposição inicial de conteúdos novos.
- `similarity_cluster`: cluster populacional de referência temporário.

- `feedback_capture`: coleta acelerada de sinais para refinar perfil.

📊 Regras Técnicas

- **Para usuários novos**:
  - Usar baseline populacional + perfil inicial (idade, localização, preferências).
  - Explorar com maior diversidade de recomendações no início.
  - Ajustar rapidamente via feedback dos primeiros 7 dias.
- **Para conteúdos novos**:
  - Expor para grupo de teste (amostragem de usuários com interesse similar).
  - Recalibrar exposição com base no engajamento real (CTR, tempo de leitura, feedback).
  - Evitar saturação: limitar impulsionamento até validação mínima.

🔧 Algoritmo Híbrido (Simplificado)

```
bootstrap_score(user, content) =
  0.5 * population_baseline_match
+ 0.3 * metadata_similarity
+ 0.2 * exploration_factor
```

- `population_baseline_match`: média de comportamento do cluster semelhante.
- `metadata_similarity`: comparação com interesses declarados ou tags.
- `exploration_factor`: aumenta diversidade em novos usuários/conteúdos.

🔧 Exemplo de Contrato de API

Requisição (novo usuário)

`POST /bootstrap/user`

```
{
  "user_id": "uuid_new",
```

```
  "new_user_profile": {"idade": 24, "preferencias": ["música", "arte"], "cida
de": "São Paulo"}
}
```


Resposta


```
{
  "bootstrap_recommendations": [
    {"type": "evento", "id": "event_123", "why_explanation": "Evento cultural
popular em SP"},
    {"type": "pessoa", "id": "user_456", "why_explanation": "Usuário novo co
m interesse em música"}
  ],
  "similarity_cluster": "latam_young_culture"
}
```


Requisição (conteúdo novo)

`POST /bootstrap/content`


```
{
  "content_id": "post_999",
  "new_content_metadata": {"tags": ["bem-estar"], "author": "user_321"}
}
```


Resposta


```
{
  "bootstrap_visibility": "moderada",
  "initial_audience": ["user_123", "user_456", "user_789"],
  "feedback_capture": "ativo"
}
```

🧪 Métricas de Observabilidade

- `new_user_retention_day7`: % de usuários ativos após 7 dias.
- `bootstrap_ctr`: taxa de cliques em recomendações iniciais.
- `new_content_success_rate`: % de conteúdos novos que atingem engajamento mínimo.
- `cluster_switch_rate`: % de usuários que mudam de cluster inicial após mais dados.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Usuário ou conteúdo novo] --> B[Coleta perfil básico/metadados]
    B --> C[Associa a cluster populacional temporário]
    C --> D[Gera recomendações/bootstrap_score]
    D --> E[Entrega recomendações iniciais]
    E --> F[Captura feedback acelerado]
    F --> G[Recalibra cluster e perfil]
```

✨ Fechamento da Camada

A **Camada 37** é o **ponto de partida inteligente**: resolve a falta de dados sem deixar o usuário perdido nem o conteúdo invisível.

Ela cria uma experiência inicial calorosa e relevante, transformando novos perfis e novos conteúdos em **parte viva do ecossistema desde o primeiro dia**.

📦 CAMADA 38 — Privacidade Diferencial (Amostragem)

🎯 Objetivo da Camada

Implementar técnicas de **privacidade diferencial** para proteger dados dos usuários ao mesmo tempo em que a IA Aurah Kosmos aprende com o coletivo.

A camada garante que **nenhum dado individual** possa ser identificado ou reconstruído, mesmo quando logs e métricas são compartilhados para análise.

🔑 Entradas

- `deep_logs`: registros anonimizados (Camada 19).
- `interaction_samples`: amostras de comportamento por grupo.
- `event_metrics`: engajamento, retenção, feedback.
- `noise_parameters`: parâmetros de ruído calibrados (`epsilon`, `delta`).

📤 Saídas

- `differentially_private_logs`: logs com ruído matemático aplicado.
- `sampled_metrics`: métricas coletivas seguras.
- `group_analytics`: análises de grupos, nunca de indivíduos.
- `audit_proofs`: provas formais de privacidade diferencial aplicadas.

📊 Regras Técnicas

- **Amostragem aleatória**: só parte dos dados é usada, reduzindo risco de identificação.
- **Ruído controlado**: adição de valores artificiais que distorcem levemente resultados sem comprometer insights globais.
- **Proteção de outliers**: valores extremos recebem ruído maior.
- **Auditoria obrigatória**: logs de privacidade diferencial ficam disponíveis para revisão regulatória (LGPD/GDPR).

🔧 Fórmula Simplificada

```
metric_noisy = true_metric + Laplace(0, sensitivity/epsilon)
```

- `sensitivity`: variação máxima que um único usuário pode causar.
- `epsilon`: parâmetro de privacidade (quanto menor, maior a proteção).
- `Laplace`: distribuição estatística usada para ruído.

🔧 Exemplo de Contrato de API

Requisição

`POST /privacy/sample`

```
{
  "metric": "avg_session_time",
  "true_value": 153,
  "epsilon": 0.5,
  "sensitivity": 10
}
```

Resposta

```
{
  "metric_noisy": 149,
  "audit_proofs": {
    "epsilon": 0.5,
    "delta": 1e-5,
    "sensitivity": 10
  }
}
```

🧪 Métricas de Observabilidade

- `epsilon_effective`: nível real de privacidade aplicado.
- `utility_loss`: perda percentual de precisão nos dados.
- `sample_size`: nº de usuários incluídos por análise.
- `audit_failures`: falhas detectadas em provas de privacidade.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Logs Anonimizados] --> B[Amostragem aleatória]
    B --> C[Aplicação de ruído diferencial]
    C --> D[Métricas seguras (coletivas)]
```

```
    D --> E[Relatórios + auditoria regulatória]
```

✨ Fechamento da Camada

A **Camada 38** é a **barreira invisível de confiança**.

Ela garante que a IA Aurah Kosmos aprenda com **tendências coletivas**, mas nunca viole a individualidade de cada usuário.

Assim, o FriendApp avança em inteligência sem nunca comprometer a ética ou a privacidade.

📦 CAMADA 39 — Interpretabilidade da IA (Explicações ao Usuário)

🎯 Objetivo da Camada

Garantir que todas as decisões da IA Aurah Kosmos sejam **explicáveis e transparentes** para o usuário.

O objetivo é que ninguém veja uma sugestão sem entender **"por que estou vendo isso?"**, fortalecendo a confiança e evitando a sensação de manipulação.

🔑 Entradas

- `recommendation_plan`: plano de recomendações (Camada 21).
- `feed_rank`: score de priorização de posts (Camada 08).
- `match_score`: afinidade social (Camada 06).
- `event_fit`: adequação a eventos e locais (Camada 10).
- `trust_score`: reputação do usuário (Camada 35).
- `context_vector`: ambiente atual (Camada 07).

📤 Saídas

- `why_explanation`: justificativa curta e legível ("Sugerimos este evento porque você mostrou interesse em música e está próximo do local").

- `explanation_log`: registro auditável das justificativas dadas.
- `feedback_options`: botões de resposta rápida (“foi útil?”, “não faz sentido”).
- `user_feedback_signal`: insumo para refinamento de modelos.

📊 Regras Técnicas

- **Clareza**: explicações devem ter no máximo 2 linhas, sem jargão técnico.
- **Auditabilidade**: logs de explicação devem ser salvos para revisão posterior.
- **Feedback imediato**: cada explicação deve oferecer botão para correção do modelo.
- **Opt-out**: usuário pode escolher não receber explicações, mas sempre terá opção.
- **Consistência**: toda recomendação deve vir acompanhada de justificativa.

🔧 Fórmula Simplificada

```
why_explanation = f(match_score, event_fit, context_vector, trust_score)
```

Exemplo:

- Se `match_score > 0.8` → “Vocês têm alta compatibilidade baseada em interesses comuns.”
- Se `event_fit > 0.7` → “Este evento combina com seu perfil e está acontecendo perto de você.”

🔧 Exemplo de Contrato de API

Requisição

`GET /recommendations/explain?user_id=uuid&item_id=event_123`

Resposta

```
{
  "item_id": "event_123",
  "why_explanation": "Evento de arte recomendado porque você curtiu post
s semelhantes e está próximo do local",
```

```
  "explanation_log": {
    "timestamp": 1726582000,
    "factors": {
      "event_fit": 0.81,
      "context_alignment": 0.73,
      "interest_overlap": 0.88
    }
  }
}
```

🧪 Métricas de Observabilidade

- `explanation_view_rate`: % de usuários que visualizaram justificativas.
- `feedback_response_rate`: % de usuários que responderam com feedback.
- `useful_explanation_rate`: % de explicações marcadas como úteis.
- `model_adjustment_count`: nº de ajustes feitos com base no feedback.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[IA gera recomendação] --> B[Calcula fatores de decisão]
    B --> C[Gera why_explanation]
    C --> D[Exibe recomendação + justificativa]
    D --> E[Captura feedback do usuário]
    E --> F[Atualiza modelos + explanation_log]
```

✨ Fechamento da Camada

A **Camada 39** transforma a IA Aurah Kosmos em uma **parceira transparente**.

Cada sugestão vem acompanhada de uma explicação clara, auditável e ajustável, fortalecendo a confiança do usuário.

Aqui, a inteligência deixa de ser uma “caixa-preta” e se torna **um sistema interpretável e colaborativo**.

📦 CAMADA 40 — Limites & Segurança Psicológica

🎯 Objetivo da Camada

Definir e aplicar **limites claros para a atuação da IA Aurah Kosmos**, prevenindo que ela seja invasiva ou cause sobrecarga emocional.

Essa camada garante que o FriendApp seja **acolhedor e seguro**, respeitando sempre o espaço mental e o bem-estar do usuário.

🔑 Entradas

- `emotional_state_score`: estado emocional atual (Camada 04).
- `collapse_risk`: risco de colapso calculado (Camada 67).
- `notification_volume`: número de notificações já enviadas (Camada 31).
- `ai_proactive_actions`: contagem de ações proativas da IA.
- `user_feedback_signals`: feedback direto do usuário (“foi útil?” / “estou sobrecarregado”).

📤 Saídas

- `proactivity_limit_flag`: indicador se a IA deve reduzir ou pausar ações proativas.
- `safe_mode_status`: ativação de modo neutro em casos críticos.
- `cooldown_timer`: período mínimo antes de nova sugestão.
- `escalation_trigger`: encaminhamento para revisão humana em casos graves.

📊 Regras Técnicas

- **Proatividade limitada**: a IA só pode realizar até 3 ações proativas por dia por usuário.
- **Cooldown obrigatório**: após uma ação proativa, deve esperar pelo menos 2h antes da próxima.
- **Safe Mode**: se `collapse_risk ≥ 0.8`, a IA entra em modo neutro, evitando estímulos adicionais.

- **Feedback imediato**: se usuário sinalizar sobrecarga, IA pausa ações por 24h.
- **Escalonamento humano**: casos persistentes de retração emocional são revisados por equipe.

🔧 Fórmula Simplificada

```
if ai_proactive_actions >= 3 or notification_volume >= 3:
    proactivity_limit_flag = true

if collapse_risk >= 0.8:
    safe_mode_status = true
```

🔧 Exemplo de Contrato de API

Requisição

`POST /safety/limits`

```
{
  "user_id": "uuid",
  "emotional_state_score": 0.22,
  "collapse_risk": 0.85,
  "ai_proactive_actions": 4,
  "notification_volume": 5
}
```

Resposta

```
{
  "proactivity_limit_flag": true,
  "safe_mode_status": true,
  "cooldown_timer": 7200,
  "escalation_trigger": true}
```

🧪 Métricas de Observabilidade

- `avg_proactive_actions_per_user`: média de ações proativas por usuário/dia.
- `safe_mode_activation_rate`: % de sessões que entraram em modo neutro.
- `cooldown_trigger_rate`: frequência de pausas aplicadas.
- `user_overload_feedback_rate`: % de usuários que relataram sobrecarga.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Captura estado emocional + risco + volume de IA] --> B[Calcula thresholds]
    B --> C{Limites ultrapassados?}
    C -- Sim --> D[Ativa proactivity_limit_flag]
    D --> E{Risco alto?}
    E -- Sim --> F[Safe Mode + escalonamento humano]
    E -- Não --> G[Aplicar cooldown]
    C -- Não --> H[Operação normal]
```

✨ Fechamento da Camada

A **Camada 40** é o **freio de segurança** do FriendApp.

Ela impede que a IA ultrapasse limites emocionais, mantendo o equilíbrio entre suporte e autonomia do usuário.

Aqui, o cuidado é explícito: a tecnologia **sabe até onde pode ir e quando precisa parar**.

📦 CAMADA 41 — Arquitetura MVP (Monólito Modular)

🎯 Objetivo da Camada

Definir a **arquitetura mínima viável (MVP)** da IA Aurah Kosmos e do FriendApp, implementada como um **monólito modular**.

O foco é entregar rapidamente um sistema funcional, simples de manter e evoluir, sem a complexidade inicial de dezenas de microserviços.

🔑 Entradas

- `user_requests`: requisições dos usuários (login, feed, chat, eventos).
- `session_data`: contexto de uso.
- `recommendation_inputs`: vetores e métricas gerados em camadas anteriores.
- `safety_flags`: indicadores de risco (Camada 15).
- `billing_events`: eventos de monetização (Camada 14).

📤 Saídas

- `responses`: respostas diretas para cada rota (feed, chat, recomendações).
- `logs`: registros de telemetria e segurança (Camada 18/19).
- `db_updates`: persistência em banco de dados único com schemas isolados.
- `ws_streams`: streams de dados em tempo real via WebSocket.

📊 Regras Técnicas

- **Camadas lógicas internas**: um único processo dividido em módulos (auth, feed, chat, planner, safety).
- **Comunicação simplificada**:
  - REST API para comandos síncronos.
  - WebSocket para eventos em tempo real (ex.: chat, notificações).
- **Banco centralizado**: PostgreSQL (estruturado) + Firestore (eventos rápidos) + Redis (cache).
- **Deploy único**: aplicação empacotada em container (Docker/Kubernetes).
- **Observabilidade integrada**: métricas e logs coletados nativamente, sem fila distribuída no MVP.

🔧 Exemplo de Estrutura Modular

```
aurah-core (monólito)
 ├── auth_module
 ├── profile_module
 ├── feed_module
 ├── chat_module
 ├── planner_module
 ├── safety_module
 ├── billing_module
 └── logging_module
```

🔧 Protocolos

- REST → endpoints de cadastro, perfil, feed, eventos, billing.
- WebSocket → chat, notificações em tempo real, check-ins.
- Kafka/Streams → **não usados no MVP**, só planejados para fase futura (Camada 42).

🔧 Exemplo de Endpoint

`GET /feed?user_id=uuid`

```
[
  {
    "post_id": "123",
    "feed_rank": 0.87,
    "why_explanation": "Compatibilidade alta + evento próximo"
  }
]
```

🧪 Métricas de Observabilidade

- `request_latency_ms`: tempo de resposta médio por endpoint.
- `ws_connection_stability`: % de conexões WebSocket estáveis.
- `error_rate`: % de requisições com erro.

- `cpu_mem_usage`: uso de CPU/memória do container.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Usuário envia requisição] --> B[aurah-core monólito]
    B --> C[Auth Module valida sessão]
    C --> D[Encaminha ao módulo correto]
    D --> E[Feed/Chat/Planner/Safety]
    E --> F[Resposta REST ou WS]
    E --> G[Persistência no banco central]
    G --> H[Logs + métricas de telemetria]
```

✨ Fechamento da Camada

A **Camada 41** define a arquitetura inicial do FriendApp como um **monólito modular simples e escalável**.

Esse design permite entregar rapidamente o MVP, com rotas claras e módulos isolados dentro de um único serviço.

Aqui, o foco é **executar com baixo risco e alta velocidade**, sem abrir mão de segurança ou ética.

📦 CAMADA 42 — Roadmap Microserviços (Fase 2)

🎯 Objetivo da Camada

Definir o **plano de evolução da arquitetura** do FriendApp, migrando do **monólito modular (Camada 41)** para uma **arquitetura de microserviços distribuídos** conforme o sistema escalar.

Essa camada garante **resiliência, escalabilidade e autonomia dos módulos críticos**, sem comprometer simplicidade no início.

🔑 Entradas

- `mvp_metrics`: métricas do monólito em produção (latência, erros, tráfego).
- `bottleneck_logs`: gargalos identificados (feed, chat, moderação).
- `user_growth_rate`: taxa de crescimento da base de usuários.
- `feature_expansion`: novas features que exigem autonomia de deploy.
- `observability_reports`: relatórios de carga do sistema (Camada 18).

📤 Saídas

- `microservices_plan`: lista de serviços a serem extraídos.
- `migration_schedule`: cronograma de transição por prioridade.
- `service_contracts`: contratos de API independentes.
- `infra_changes`: ajustes de infraestrutura (clusters, filas, balanceadores).

📊 Regras Técnicas

- **Critério de extração**: um módulo só vira microserviço se for gargalo comprovado ou exigir deploy independente.
- **Ordem sugerida de extração**:
  1. **Feed Service** → alto tráfego, consultas intensivas.
  2. **Chat Service** → baixa latência, tempo real.
  3. **Safety Service** → moderação e firewall dedicados.
  4. **Logging Service** → observabilidade escalável.
- **Comunicação híbrida**:
  - REST/gRPC para requisições diretas.
  - Kafka/EventBus para eventos assíncronos (check-ins, logs, notificações).
- **Banco por serviço**: cada microserviço tem schema isolado; dados compartilhados via APIs.

🔧 Exemplo de Microserviços Planejados

```
aurah-core (restante do monólito)
 ├── feed-service
 ├── chat-service
 ├── safety-service
 ├── logging-service
 └── billing-service (opcional na fase 2)
```

🔧 Exemplo de Contrato de API (Feed Service)

Requisição

`GET /feed?user_id=uuid`

Resposta

```
[
  {
    "post_id": "123",
    "feed_rank": 0.87,
    "why_explanation": "Compatibilidade alta + evento próximo"
  }
]
```

🧪 Métricas de Observabilidade

- `migration_progress`: % de módulos já extraídos.
- `latency_delta`: diferença de latência antes/depois da migração.
- `error_rate_change`: alteração na taxa de erros pós-extração.
- `infra_cost_increase`: custo adicional de rodar microserviços.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Monólito MVP em produção] --> B[Monitorar métricas]
    B --> C[Identificar gargalos]
```

```
C --> D[Selecionar módulo para extração]
D --> E[Criar microserviço independente]
E --> F[Configurar API contracts + observabilidade]
F --> G[Deploy e monitoramento contínuo]
G --> H[Iterar até arquitetura híbrida estável]
```

✨ Fechamento da Camada

A **Camada 42** é o **plano de crescimento sustentável**.

Ela mostra como o FriendApp vai evoluir de um **monólito ágil** para uma **arquitetura distribuída e resiliente**, garantindo escalabilidade sem complexidade prematura.

Aqui, a IA Aurah Kosmos se prepara para **crescer junto com a base global de usuários**.

📦 CAMADA 43 — Data Lake & Feature Store

🎯 Objetivo da Camada

Gerenciar o **armazenamento massivo de dados (Data Lake)** e a **preparação de features reutilizáveis (Feature Store)** para treinar e operar os modelos da IA Aurah Kosmos.

Essa camada garante que dados brutos, semiestruturados e processados sejam **organizados, seguros e prontos para uso**, evitando duplicação e acelerando aprendizado.

🔑 Entradas

- `deep_logs`: registros anonimizados de interações (Camada 19).
- `feedback_logs`: avaliações explícitas de usuários (Camada 22).
- `event_metrics`: dados de eventos e check-ins (Camada 10 e 23).
- `reputation_data`: scores de confiança (Camada 35).

- `privacy_noise`: ruído aplicado pela privacidade diferencial (Camada 38).

📤 Saídas

- `data_lake_storage`: repositório central de dados brutos.
- `feature_store`: repositório de features limpas e versionadas para modelos.
- `training_datasets`: conjuntos de treino preparados.
- `model_inputs`: features em tempo real para inferência da IA.

📊 Regras Técnicas

- **Data Lake**:
  - Armazenamento em BigQuery/S3 (bruto e histórico).
  - Dados sempre anonimizados.
  - Retenção longa (2 anos) com compressão.
- **Feature Store**:
  - Features documentadas, versionadas e validadas.
  - Reuso de features entre modelos (evita duplicação).
  - Baixa latência para leitura online (inferência).
  - Compatibilidade com pipelines de MLOps (Camada 44).

🔧 Estrutura de Pastas (exemplo simplificado)

```
/data_lake
  /raw
    /logs
    /events
  /processed
    /feedback
    /reputation
/feature_store
  /v1
    /user_embeddings
    /event_fit_scores
```

```
  /v2
    /collapse_risk_features
```

🔧 Exemplo de Feature

Nome: `collapse_risk_features`

Descrição: features usadas para calcular risco de colapso (Camada 67).

Inputs: `neg_sent`, `inactivity_rate`, `report_count_norm`, `hesitation_spike`.

Output: vetor normalizado para modelos preditivos.

🔧 Exemplo de Contrato de API

Requisição (extrair features)

`GET /features/user?user_id=uuid&version=v2`

Resposta

```
{
  "user_id": "uuid",
  "features": {
    "collapse_risk_features": [0.45, 0.3, 0.2, 0.15],
    "trust_score_norm": 0.82,
    "compatibility_embedding": [0.23, -0.11, 0.84, ...]
  }
}
```

🧪 Métricas de Observabilidade

- `feature_reuse_rate`: % de features reaproveitadas entre modelos.
- `dataset_freshness`: tempo médio de atualização dos datasets.
- `error_rate_extraction`: falhas na geração de features.
- `storage_cost`: custo mensal do data lake.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Captura deep_logs + eventos + feedback] --> B[Armazena no Data Lak
e]
    B --> C[Pipeline de limpeza e anonimização]
    C --> D[Criação de features validadas]
    D --> E[Persistência na Feature Store]
    E --> F[Uso em treino de modelos e inferência em tempo real]
```

✨ Fechamento da Camada

A **Camada 43** é a **memória organizada da IA Aurah Kosmos**.

O Data Lake armazena tudo de forma bruta e segura, enquanto a Feature Store transforma dados em **ingredientes prontos para inteligência**.

Aqui, o aprendizado do FriendApp deixa de ser improvisado e passa a ser **estruturado, versionado e reutilizável**.

📦 CAMADA 44 — MLOps & Versionamento de Modelos

🎯 Objetivo da Camada

Garantir que todos os **modelos de IA da Aurah Kosmos** sejam treinados, versionados, validados e implantados de forma **confiável, auditável e escalável**.

Essa camada sustenta o ciclo de vida dos modelos, evitando regressões e garantindo que apenas versões seguras e éticas sejam usadas em produção.

🔑 Entradas

- `training_datasets`: conjuntos preparados pela Feature Store (Camada 43).
- `model_configs`: hiperparâmetros e definições de arquitetura.
- `evaluation_metrics`: métricas de validação (acurácia, F1, bias).
- `rollback_signals`: alertas de falha em produção.
- `user_feedback_signals`: feedback explícito dos usuários (Camada 22).

📤 Saídas

- `model_versions`: versões numeradas e documentadas (`AURAH-M.m.p`).
- `deployment_status`: status de cada modelo em produção.
- `rollback_logs`: registros de reversões automáticas.
- `model_registry`: catálogo de modelos ativos, estáveis e depreciados.

📊 Regras Técnicas

- **Versionamento semântico**:
  - `MAJOR`: mudança estrutural (ex.: novo embedding).
  - `MINOR`: novas habilidades sensoriais (ex.: RA, matching).
  - `PATCH`: correções finas de bug ou performance.
- **Validação obrigatória**: modelo só vai para produção se atender KPIs mínimos (ex.: acurácia > 93%, latência < 300ms, viés ≤ 5%).
- **Champion/Challenger**: sempre testar modelo novo contra modelo atual antes do rollout.
- **Rollback automático**: se métricas em produção caírem >10% em 24h, modelo volta à versão anterior.
- **Auditoria contínua**: logs e métricas disponíveis para revisão humana.

🔧 Exemplo de Estrutura de Registro

```
{
  "model_name": "aurah_emotion_v3",
  "version": "3.2.1",
  "registry_status": "active",
  "evaluation_metrics": {
    "accuracy": 0.94,
    "f1_score": 0.91,
    "bias_score": 0.03
  },
  "deployment_status": "production",
  "last_update": "2025-09-01T10:00:00Z"
```

```
}
```

🧪 Métricas de Observabilidade

- `model_accuracy_production`: acurácia real em produção.
- `rollback_count`: nº de vezes que rollback foi necessário.
- `model_drift_rate`: % de deriva nos dados vs treino.
- `bias_alert_rate`: alertas de viés disparados.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Feature Store entrega dataset] --> B[Treino do modelo]
    B --> C[Validação offline]
    C --> D{KPIs atendidos?}
    D -- Não --> E[Modelo rejeitado]
    D -- Sim --> F[Registro no model_registry]
    F --> G[Teste Champion/Challenger]
    G --> H{Melhor que atual?}
    H -- Sim --> I[Deploy gradual em produção]
    H -- Não --> J[Mantém versão anterior]
    I --> K[Monitoramento contínuo]
    K --> L{Métricas caíram?}
    L -- Sim --> M[Rollback automático]
    L -- Não --> N[Versão estabilizada]
```

✨ Fechamento da Camada

A **Camada 44** garante que a IA Aurah Kosmos não seja instável nem opaca.

Cada modelo treinado tem **história, versão e critérios de qualidade documentados**, permitindo evolução segura e transparente.

Aqui, a inteligência artificial não é improviso — é **engenharia viva, com governança e responsabilidade**.

📦 CAMADA 45 — Edge Signals (On-device)

🎯 Objetivo da Camada

Capturar **sinais leves diretamente no dispositivo do usuário** (on-device), reduzindo latência, preservando privacidade e permitindo que a IA Aurah Kosmos tenha percepções rápidas sem depender 100% da nuvem.

Essa camada equilibra **privacidade + responsividade**, com processamento local e envio apenas de metadados essenciais.

🔑 Entradas

- `device_sensors`: acelerômetro, giroscópio, luz ambiente (opt-in).
- `ui_events`: cliques, tempo de tela, hesitação (Camada 05).
- `offline_logs`: interações armazenadas sem internet.
- `battery_status`: estado de bateria (para calibrar consumo).
- `privacy_prefs`: permissões do usuário para coleta local.

📤 Saídas

- `edge_signals`: pacote de sinais processados localmente (anônimos).
- `cached_data`: dados armazenados offline para sincronização futura.
- `lightweight_inferences`: inferências rápidas (ex.: nível de atenção, movimento).
- `privacy_report`: resumo local do que foi coletado e enviado.

📊 Regras Técnicas

- **Opt-in obrigatório**: sensores só coletam dados se ativados explicitamente.
- **Edge-first**: sinais processados no próprio dispositivo → apenas metadados anonimizados são enviados.
- **Sincronização offline**: se não houver internet, dados são salvos e enviados criptografados quando conexão retornar.

- **Consumo otimizado**: coleta só acontece em baixa frequência e nunca se o dispositivo estiver com bateria < 20%.

🔧 Exemplo de Estrutura de Edge Signal

```
{
  "user_id": "uuid",
  "timestamp": 1726590000,
  "ui_events": {"clicks": 12, "avg_hesitation_ms": 850},
  "device_sensors": {"movement": "leve", "screen_brightness": 0.6},
  "battery_status": "72%",
  "privacy_flags": {"share_motion": true, "share_location": false}
}
```

🧪 Métricas de Observabilidade

- `edge_signal_volume`: nº de pacotes enviados por dia.
- `avg_latency_edge_inference`: tempo médio de inferência no dispositivo.
- `offline_sync_rate`: % de dados enviados após reconexão.
- `user_opt_in_rate`: % de usuários que ativaram coleta on-device.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Captura sensores + eventos UI] --> B[Processamento local on-device]
    B --> C[Anonimização e compressão]
    C --> D{Conexão disponível?}
    D -- Sim --> E[Envia edge_signals p/ nuvem]
    D -- Não --> F[Salva em cached_data]
    F --> G[Sincroniza quando online]
```

✨ Fechamento da Camada

A **Camada 45** dá ao FriendApp a capacidade de ser **rápido e privado ao mesmo tempo**.

O usuário sente fluidez (inferências imediatas), enquanto seus dados permanecem **sob controle e processados localmente**.

Aqui, a IA Aurah Kosmos mostra que respeita o dispositivo e a intimidade de cada pessoa.

🌍 CAMADA 46 — Internacionalização (i18n/l10n)

🎯 Objetivo da Camada

Permitir que a IA Aurah Kosmos e todo o FriendApp funcionem de forma **global e culturalmente adaptada**, respeitando **idiomas, símbolos, costumes e contextos regionais**.

Essa camada garante que cada interação seja **fluida, inclusiva e precisa**, independentemente da língua ou cultura do usuário.

🔑 Entradas

- `user_locale`: idioma, país e preferências definidas no cadastro (Camada 08).
- `device_settings`: idioma e timezone do dispositivo.
- `cultural_context`: detectado pela IA com base em expressões, emojis e padrões regionais.
- `translation_memory`: repositório de traduções humanas aprovadas.
- `ai_translation`: motores de tradução neural para fallback em tempo real.

📤 Saídas

- `localized_ui`: textos, ícones, datas e números adaptados ao idioma/região.
- `aurah_responses_localized`: respostas da IA traduzidas e adaptadas culturalmente.
- `feed_localized`: conteúdos adaptados para relevância local.
- `logs_translation`: histórico com versão original e traduzida para auditoria.

📊 Regras Técnicas

- **i18n (internacionalização)**:
  - O código do app deve estar livre de strings fixas.
  - Suporte a Unicode e RTL (Right-to-Left).
  - Datas, moedas e formatos padronizados via ISO.
- **l10n (localização)**:
  - Textos adaptados com contexto cultural (ex.: evitar termos técnicos confusos em regiões específicas).
  - Emoticons e expressões adaptadas culturalmente.
  - Filtros de conteúdo sensível (ex.: símbolos proibidos em certas regiões).
- **Fallback inteligente**:
  - Caso não exista tradução aprovada → IA Neural traduz.
  - Caso IA falhe → exibir versão em inglês com aviso “Tradução em processamento”.

🔧 Exemplo de Estrutura

```
{
  "user_id": "uuid",
  "locale": "pt-BR",
  "aurah_response": {
    "original": "Your energy is aligned with this connection.",
    "localized": "Sua energia está alinhada com esta conexão."
  },
  "ui_elements": {
    "date_format": "DD/MM/YYYY",
    "currency": "R$",
    "rtl": false}
}
```

🧪 Métricas de Observabilidade

- `translation_accuracy_score`: taxa de aceitação da tradução.

- `latency_translation`: tempo médio de resposta traduzida.
- `fallback_rate`: % de vezes que o sistema usou fallback neural.
- `user_feedback_translation`: avaliações explícitas de tradução (👍👎).

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Usuário interage com IA] --> B[Detecção de locale]
    B --> C{Existe tradução humana?}
    C -- Sim --> D[Aplica tradução aprovada]
    C -- Não --> E[Fallback: IA Neural traduz]
    D --> F[Entrega resposta localizada]
    E --> F[Entrega resposta localizada]
    F --> G[Armazena em logs_translation]
    G --> H[Feedback do usuário melhora Translation Memory]
```

✨ Fechamento da Camada

A **Camada 46** garante que o FriendApp não seja apenas traduzido, mas **culturalmente vivo em cada canto do planeta**.

Aqui, a Aurah Kosmos aprende idiomas, símbolos e nuances regionais, sempre entregando **clareza, empatia e coerência local**.

📦 CAMADA 47 — Feedback Inline do Usuário (Explainable AI)

🎯 Objetivo da Camada

Dar ao usuário o poder de **avaliar em tempo real as respostas e recomendações da IA Aurah Kosmos**, diretamente onde elas aparecem.

Essa camada fecha o ciclo de interpretabilidade: toda sugestão vem com uma explicação (Camada 39) **e uma forma imediata de dizer se aquilo foi útil ou não**.

🔑 Entradas

- `user_id`: identificador único.
- `item_id`: conteúdo ou recomendação exibida (post, evento, conexão, chat).
- `why_explanation`: justificativa associada (Camada 39).
- `feedback_signal`: resposta do usuário (👍 / 👎, “foi útil?”).
- `text_comment`: comentário opcional do usuário.

📤 Saídas

- `feedback_log`: registro da avaliação feita.
- `model_adjustment_signal`: sinal enviado para modelos recalibrarem fatores.
- `audit_log`: trilha de feedbacks salvos para auditoria.
- `user_control_ui`: exibição imediata de que feedback foi processado.

📊 Regras Técnicas

- **Simplicidade absoluta**: botões claros (“foi útil?” / “não fez sentido”).
- **Não intrusivo**: feedback aparece como opção, nunca como obrigação.
- **Peso real**: feedback explícito tem prioridade sobre inferências do modelo.
- **Transparência**: usuário pode ver histórico de feedbacks dados.
- **Ciclo fechado**: feedback deve impactar recomendações futuras.

🔧 Exemplo de Contrato de API

Requisição

`POST /feedback/inline`

```
{
  "user_id": "uuid",
  "item_id": "event_123",
  "feedback_signal": "not_useful",
  "text_comment": "Esse evento não tem nada a ver comigo"
}
```

Resposta

```
{
  "status": "ok",
  "feedback_log": {
    "item_id": "event_123",
    "signal": "not_useful",
    "timestamp": 1726598000
  },
  "model_adjustment_signal": true}
```

🧪 Métricas de Observabilidade

- `feedback_response_rate`: % de usuários que interagem com feedbacks.
- `useful_rate`: % de feedbacks positivos.
- `adjustment_latency`: tempo entre feedback e impacto percebido em recomendações.
- `false_feedback_rate`: % de feedbacks incoerentes ou maliciosos.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Recomendação exibida com why_explanation] --> B[Usuário escolhe 👍 ou 👎]
    B --> C[Registro em feedback_log]
    C --> D[Envio para modelos como adjustment_signal]
    D --> E[Atualização em recomendações futuras]
    C --> F[Armazenamento em audit_log]
```

✨ Fechamento da Camada

A **Camada 47** fecha o ciclo da **IA explicável e colaborativa**.

Aqui, o usuário não apenas entende o motivo de cada sugestão, mas também **corrige a IA em tempo real**, moldando sua própria experiência.

É o momento em que o FriendApp deixa de ser só um app e se torna **uma parceria viva entre inteligência artificial e inteligência humana**.

📦 CAMADA 48 — Auditoria & Compliance

🎯 Objetivo da Camada

Garantir que todas as ações do FriendApp e da IA Aurah Kosmos sejam **auditáveis, conformes às legislações (LGPD, GDPR, CCPA)** e revisáveis por equipes humanas ou órgãos externos.

Essa camada cria os mecanismos que dão **confiança institucional**, mostrando que o ecossistema não é apenas seguro para usuários, mas também **legalmente sólido e transparente**.

🔑 Entradas

- `consent_audit_trail`: histórico de consentimentos (Camada 20).
- `moderation_logs`: registros de ações de segurança/moderação (Camada 15 e 29).
- `privacy_differential_logs`: métricas com ruído aplicado (Camada 38).
- `transaction_logs`: histórico econômico (Camada 14 e 27).
- `model_versions`: versões da IA em produção (Camada 44).

📤 Saídas

- `compliance_reports`: relatórios periódicos para reguladores.
- `audit_hashes`: registros imutáveis (hash criptográfico).
- `external_audit_access`: APIs seguras para auditorias externas.
- `violation_flags`: alertas de violações ou riscos éticos.

📊 Regras Técnicas

- **Logs imutáveis**: todos os registros críticos são salvos com hash e timestamp.

- **Relatórios periódicos**: geração automática mensal de relatórios de conformidade.
- **Auditoria externa**: APIs permitem auditores externos verificarem amostras em tempo real.
- **Detecção de violações**: triggers automáticas em caso de descumprimento de políticas.
- **Transparência ao usuário**: painel simplificado mostra histórico de ações e consentimentos.

🔧 Exemplo de Registro de Auditoria

```
{
  "log_id": "audit_123",
  "timestamp": 1726602000,
  "action": "post_removal",
  "reason": "linguagem_de_ódio",
  "hash": "abc123xyz",
  "verified_by": "IA + humano",
  "status": "compliant"
}
```

🔧 Exemplo de Relatório de Compliance

Requisição

`GET /compliance/report?period=2025-08`

Resposta

```
{
  "period": "2025-08",
  "moderation_actions": 1520,
  "privacy_audits": 98,
  "user_consent_updates": 430,
  "model_versions_active": ["aurah_emotion_v3.2.1"],
  "violations_detected": 0
```

```
}
```

🧪 Métricas de Observabilidade

- `audit_log_growth`: crescimento dos registros de auditoria.
- `external_audit_requests`: nº de acessos de auditores externos.
- `violation_incidents`: nº de violações detectadas.
- `compliance_score`: índice interno de conformidade (0–100).

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Captura logs de ações críticas] --> B[Gera hashes imutáveis]
    B --> C[Armazena em audit_log]
    C --> D[Relatórios mensais automáticos]
    D --> E[APIs para auditoria externa]
    E --> F{Violação detectada?}
    F -- Sim --> G[Dispara violation_flag + revisão]
    F -- Não --> H[Sistema permanece compliant]
```

✨ Fechamento da Camada

A **Camada 48** é o **escudo institucional do FriendApp**.

Ela garante que o sistema esteja sempre **alinhado às leis e padrões internacionais de proteção de dados**, com rastreabilidade absoluta e possibilidade de auditoria independente.

Aqui, o compromisso com **ética e legalidade** é tão forte quanto o compromisso com conexões autênticas.

📦 CAMADA 48 — Auditoria & Compliance

🎯 Objetivo da Camada

Garantir que todas as ações do FriendApp e da IA Aurah Kosmos sejam **auditáveis, conformes às legislações (LGPD, GDPR, CCPA)** e revisáveis por equipes humanas ou órgãos externos.

Essa camada cria os mecanismos que dão **confiança institucional**, mostrando que o ecossistema não é apenas seguro para usuários, mas também **legalmente sólido e transparente**.

🔑 Entradas

- `consent_audit_trail`: histórico de consentimentos (Camada 20).
- `moderation_logs`: registros de ações de segurança/moderação (Camada 15 e 29).
- `privacy_differential_logs`: métricas com ruído aplicado (Camada 38).
- `transaction_logs`: histórico econômico (Camada 14 e 27).
- `model_versions`: versões da IA em produção (Camada 44).

📤 Saídas

- `compliance_reports`: relatórios periódicos para reguladores.
- `audit_hashes`: registros imutáveis (hash criptográfico).
- `external_audit_access`: APIs seguras para auditorias externas.
- `violation_flags`: alertas de violações ou riscos éticos.

📊 Regras Técnicas

- **Logs imutáveis**: todos os registros críticos são salvos com hash e timestamp.
- **Relatórios periódicos**: geração automática mensal de relatórios de conformidade.
- **Auditoria externa**: APIs permitem auditores externos verificarem amostras em tempo real.
- **Detecção de violações**: triggers automáticas em caso de descumprimento de políticas.
- **Transparência ao usuário**: painel simplificado mostra histórico de ações e consentimentos.

🔧 Exemplo de Registro de Auditoria

```
{
  "log_id": "audit_123",
  "timestamp": 1726602000,
  "action": "post_removal",
  "reason": "linguagem_de_ódio",
  "hash": "abc123xyz",
  "verified_by": "IA + humano",
  "status": "compliant"
}
```

🔧 Exemplo de Relatório de Compliance

Requisição

`GET /compliance/report?period=2025-08`

Resposta

```
{
  "period": "2025-08",
  "moderation_actions": 1520,
  "privacy_audits": 98,
  "user_consent_updates": 430,
  "model_versions_active": ["aurah_emotion_v3.2.1"],
  "violations_detected": 0
}
```

🧪 Métricas de Observabilidade

- `audit_log_growth`: crescimento dos registros de auditoria.
- `external_audit_requests`: nº de acessos de auditores externos.
- `violation_incidents`: nº de violações detectadas.
- `compliance_score`: índice interno de conformidade (0–100).

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Captura logs de ações críticas] --> B[Gera hashes imutáveis]
    B --> C[Armazena em audit_log]
    C --> D[Relatórios mensais automáticos]
    D --> E[APIs para auditoria externa]
    E --> F{Violação detectada?}
    F -- Sim --> G[Dispara violation_flag + revisão]
    F -- Não --> H[Sistema permanece compliant]
```

✨ Fechamento da Camada

A **Camada 48** é o **escudo institucional do FriendApp**.

Ela garante que o sistema esteja sempre **alinhado às leis e padrões internacionais de proteção de dados**, com rastreabilidade absoluta e possibilidade de auditoria independente.

Aqui, o compromisso com **ética e legalidade** é tão forte quanto o compromisso com conexões autênticas.

📦 CAMADA 50 — Performance & Latência

🎯 Objetivo da Camada

Garantir que todas as funcionalidades do FriendApp e da IA Aurah Kosmos operem com **resposta rápida, estável e previsível**, mantendo a experiência fluida mesmo em cenários de alta carga.

Essa camada estabelece métricas de **latência alvo, limites de consumo e monitoramento contínuo**.

🔑 Entradas

- `request_logs`: registros de chamadas API.

- `ws_events`: eventos de WebSocket (chat, notificações).
- `db_queries`: consultas ao PostgreSQL/Firestore.
- `cache_signals`: respostas vindas do Redis.
- `resource_usage`: métricas de CPU, memória, rede.

📤 Saídas

- `latency_report`: tempos de resposta médios e p95/p99.
- `error_alerts`: alertas em caso de degradação de performance.
- `autoscale_signals`: triggers para aumentar recursos (Kubernetes/HPA).
- `perf_score`: índice consolidado de desempenho da release.

📊 Regras Técnicas

- **Latência alvo**:
  - API crítica → p95 < 300ms.
  - Chat (WS) → latência média < 100ms.
  - Feed → resposta < 500ms para top-20 itens.
- **Uso de cache**:
  - Consultas repetitivas (feed, reputação) devem sempre passar por Redis.
  - TTL configurado entre 30s e 5min conforme endpoint.
- **Autoscala**:
  - Quando CPU > 75% ou latência p95 > 400ms por 5 min → disparar autoscaling.
- **Monitoramento**:
  - Dashboards Prometheus/Grafana em tempo real.
  - Logs críticos integrados ao painel DevOps (Camada 56).

🔧 Exemplo de Relatório de Performance

```
{
  "endpoint": "/feed",
  "requests": 182300,
  "latency": {
    "avg_ms": 240,
    "p95_ms": 310,
    "p99_ms": 480
  },
  "error_rate": 0.004,
  "autoscale_triggered": true}
```


🧪 Métricas de Observabilidade

- `avg_latency_ms`: tempo médio por endpoint.
- `p95_latency_ms`: latência em 95% das requisições.
- `p99_latency_ms`: latência em 99% das requisições.
- `error_rate`: % de falhas.
- `autoscale_count`: nº de autoscalings disparados.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Logs de requests + eventos WS] --> B[Calcula métricas de latência]
    B --> C{p95 > 300ms ou CPU > 75%?}
    C -- Sim --> D[Dispara autoscale]
    C -- Não --> E[Opera normal]
    D --> F[Atualiza dashboards e relatórios]
    E --> F
```

✨ Fechamento da Camada

A **Camada 50** é o **termômetro de velocidade do FriendApp**.

Ela garante que o sistema seja sempre rápido, estável e previsível, mesmo em cenários de picos.

Aqui, performance não é luxo — é **pré-requisito para confiança e pertencimento**.

📦 CAMADA 51 — Observabilidade Emocional Coletiva

🎯 Objetivo da Camada

Monitorar em tempo real as **ondas emocionais coletivas** dentro do FriendApp, detectando **picos de expansão ou retração** em regiões, grupos e eventos.

Essa camada permite que a IA Aurah Kosmos **perceba o coletivo como organismo vivo**, ajudando a prevenir colapsos sociais e a promover experiências mais harmônicas.

🔑 Entradas

- `aggregate_emotional_state`: estado médio das células do mapa (Camada 12).
- `checkin_logs`: presença confirmada em eventos e locais (Camada 23).
- `feedback_signals`: avaliações pós-interação (Camada 22).
- `event_metadata`: informações sobre eventos em andamento (Camada 10).
- `context_vector`: dados temporais e geográficos (Camada 07).

📤 Saídas

- `collective_wave_index`: índice de expansão/retração em cada região.
- `collective_alerts`: alertas disparados em caso de anomalias (colapso coletivo).
- `visual_heatmaps`: representações visuais das ondas no Mapa de Frequência.
- `trend_reports`: relatórios de tendências (ex.: crescimento de conexões em uma cidade).

📊 Regras Técnicas

- **Anonimização obrigatória**: todos os cálculos são feitos em clusters, nunca em indivíduos.
- **Janelas temporais**: agregação em blocos de 15min, 1h e 24h.
- **Sensibilidade adaptativa**: regiões com muitos usuários têm thresholds mais altos para disparo de alerta.
- **Alertas inteligentes**: IA só envia alertas se padrões persistirem >30min.
- **Auditoria**: logs das ondas coletivas são armazenados para análise posterior.

🔧 Fórmula Simplificada

```
collective_wave_index =
  (Δ aggregate_emotional_state / Δ tempo)
 * normalização(cluster_size)
```

- Valores positivos altos → expansão coletiva.
- Valores negativos altos → retração ou colapso.

🔧 Exemplo de Contrato de API

Requisição

`GET /collective/state?geo_hash=6gkz7`

Resposta

```
{
  "geo_hash": "6gkz7",
  "collective_wave_index": -0.42,
  "collective_alerts": ["Risco de retração detectado nesta região"],
  "visual_heatmaps": {
    "color": "#FF6A6A",
    "intensity": "alta"
  },
  "trend_reports": [
    {"period": "1h", "trend": "queda"},
    {"period": "24h", "trend": "estável"}
```

```
  ]
}
```

🧪 Métricas de Observabilidade

- `wave_alerts_triggered`: nº de alertas coletivos disparados.
- `expansion_zone_count`: quantidade de regiões em expansão.
- `collapse_zone_count`: nº de regiões em retração.
- `accuracy_feedback_vs_wave`: correlação entre feedbacks individuais e ondas detectadas.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Captura estados agregados do mapa] --> B[Agregação temporal por cl
uster]
    B --> C[Calcula collective_wave_index]
    C --> D{Anomalia persistente?}
    D -- Sim --> E[Dispara collective_alert]
    D -- Não --> F[Status normal]
    E --> G[Atualiza visual_heatmaps]
    F --> G
```

✨ Fechamento da Camada

A **Camada 51** é o **sensor coletivo da IA Aurah Kosmos**.

Ela transforma sinais dispersos em **ondas emocionais legíveis**, ajudando a identificar momentos de crise ou expansão em tempo real.

Aqui, o FriendApp se torna mais que uma rede: ele passa a ser **um barômetro vivo das emoções coletivas**.

📦 CAMADA 52 — Consentimento Granular

🎯 Objetivo da Camada

Permitir que o usuário tenha **controle total e detalhado** sobre quais dados e recursos deseja compartilhar com o FriendApp e a IA Aurah Kosmos.

Essa camada garante que cada aspecto da experiência seja **opt-in**, configurável e reversível em tempo real, fortalecendo a transparência e a confiança.

🔑 Entradas

- `user_id`: identificador único.
- `consent_flags`: conjunto de flags individuais de consentimento.
- `privacy_prefs`: preferências de privacidade definidas no cadastro (Camada 02).
- `feature_toggles`: switches ativados/desativados pelo usuário.
- `audit_logs`: histórico de mudanças anteriores em consentimentos.

📤 Saídas

- `updated_consent_flags`: novo estado granular aplicado.
- `consent_audit_trail`: trilha de auditoria com hash e timestamp.
- `feature_access_matrix`: matriz de recursos ativos/inativos por usuário.
- `user_ui_controls`: interface de controle exibida no app.

📊 Regras Técnicas

- **Granularidade máxima**: cada recurso sensível deve ter flag independente:
  - Localização precisa (geo_hash).
  - Ativação da IA proativa.
  - Armazenamento de check-ins vibracionais.
  - Notificações push.

    - Edge signals (Camada 45).
- **Efeito imediato**: mudanças aplicadas em tempo real.
- **Auditoria imutável**: toda alteração gera hash + registro em trilha de consentimento.
- **Opt-out respeitado**: se desativado, recurso deve parar imediatamente.
- **Interface clara**: UI deve exibir switches simples (ligar/desligar).

🔧 Exemplo de Contrato de API

Requisição (alterar consentimento)

`POST /consent/granular/update`

```
{
  "user_id": "uuid",
  "consent_flags": {
    "share_location": false,
    "ai_proactivity": true,
    "checkin_vibrational": false,
    "notifications_push": true}
}
```

Resposta

```
{
  "updated_consent_flags": {
    "share_location": false,
    "ai_proactivity": true,
    "checkin_vibrational": false,
    "notifications_push": true},
  "consent_audit_trail": [
    {"flag": "share_location", "status": false, "timestamp": 1726612000, "has
h": "consent123"}
  ]
}
```

🧪 Métricas de Observabilidade

- `opt_in_rate_per_feature`: % de usuários que ativaram cada recurso.
- `opt_out_reversal_rate`: nº de vezes que usuários desligaram/ligaram recursos.
- `consent_change_latency`: tempo médio entre ajuste e efeito aplicado.
- `audit_trail_integrity`: consistência e verificabilidade dos registros.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Usuário acessa painel de consentimento] --> B[Altera switches de rec
ursos]
    B --> C[Atualiza consent_flags em tempo real]
    C --> D[Persistência em consent_audit_trail com hash]
    D --> E[Atualiza feature_access_matrix]
    E --> F[Aplica ajustes imediatos em runtime]
```

✨ Fechamento da Camada

A **Camada 52** é o **painel de controle da confiança**.

Aqui, cada usuário decide exatamente até onde quer ir com a IA Aurah Kosmos, recurso por recurso, sem zonas cinzentas.

É a garantia de que o FriendApp **respeita a vontade individual acima de qualquer algoritmo**.

📦 CAMADA 53 — Retreinamento Contínuo (Micro-Ajustes)

🎯 Objetivo da Camada

Manter os modelos da IA Aurah Kosmos **sempre atualizados e calibrados** com base em novos dados, feedback dos usuários e mudanças de contexto.

Essa camada garante que a IA não fique estagnada, aprendendo continuamente de forma **segura, incremental e auditável**.

🔑 Entradas

- `feedback_logs`: feedbacks explícitos dos usuários (Camada 22 e 47).
- `interaction_scores`: métricas de qualidade de interações.
- `collapse_risk_data`: indicadores de risco (Camada 67).
- `new_features`: features atualizadas da Feature Store (Camada 43).
- `model_performance_metrics`: métricas de modelos em produção (Camada 44).

📤 Saídas

- `weight_updates`: microajustes em parâmetros dos modelos.
- `drift_alerts`: alertas de desvio de dados (data drift).
- `updated_models`: versões de modelos incrementais (patch).
- `training_logs`: registros auditáveis de retreinamento.

📊 Regras Técnicas

- **Micro-ajustes diários**: pequenas atualizações de pesos com base em feedback.
- **Retreinamento semanal**: ajustes modulares em embeddings e classificadores.
- **Treino profundo mensal**: revalidação completa dos modelos com novos datasets.
- **Limites de drift**: se modelos desviarem >5% das métricas esperadas, retreinamento é disparado automaticamente.
- **Auditoria contínua**: todos os treinos geram logs versionados (Camada 48).

🔧 Algoritmo Simplificado

```
if drift_index > threshold or feedback_negative_rate > X%:
    trigger_retrain(model_id, dataset_version)
```

🔧 Exemplo de Registro de Retreinamento

```
{
  "model_id": "aurah_emotion_v3",
  "previous_version": "3.2.1",
  "new_version": "3.2.2",
  "trigger_reason": "drift > 5%",
  "dataset_version": "v2025_09_10",
  "accuracy_before": 0.91,
  "accuracy_after": 0.94,
  "timestamp": 1726618000
}
```


🧪 Métricas de Observabilidade

- `drift_index`: grau de desvio entre dados de treino e produção.
- `feedback_alignment`: % de feedbacks positivos após ajustes.
- `model_accuracy_delta`: diferença de acurácia antes/depois do treino.
- `retrain_frequency`: nº de retreinamentos realizados por mês.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Captura feedback + métricas de produção] --> B[Calcula drift_index]
    B --> C{Drift > threshold?}
    C -- Sim --> D[Dispara retreinamento]
    D --> E[Atualiza pesos e gera versão incremental]
    E --> F[Valida resultados em champion/challenger]
    F --> G{KPIs atingidos?}
    G -- Sim --> H[Deploy em produção]
    G -- Não --> I[Rollback automático]
```

✨ Fechamento da Camada

A **Camada 53** é o mecanismo que garante que a IA Aurah Kosmos **nunca pare de evoluir**.

Ela aprende com cada interação, ajusta seus modelos de forma incremental e mantém sua **precisão, relevância e ética** ao longo do tempo.

Aqui, a IA se torna verdadeiramente **adaptativa e viva**.

📦 CAMADA 54 — Personalização Regional Profunda

🎯 Objetivo da Camada

Aprimorar a experiência do usuário adaptando o FriendApp e a IA Aurah Kosmos para **contextos regionais específicos**, indo além de tradução (Camada 46).

Aqui, entram ajustes em **hábitos sociais, preferências culturais, calendários locais, normas de convivência e sensibilidades éticas**.

O FriendApp deixa de ser apenas global para ser **localmente relevante e culturalmente inteligente**.

🔑 Entradas

- `user_locale`: idioma/região do usuário (Camada 46).
- `cultural_cluster_id`: cluster sociocultural atribuído pela IA.
- `regional_events`: calendário de datas e eventos culturais locais.
- `usage_patterns`: padrões de engajamento por região (ex.: uso maior à noite em certas culturas).
- `feedback_local`: respostas explícitas de usuários sobre adaptação cultural.

📤 Saídas

- `localized_behavioral_models`: modelos de recomendação ajustados para clusters regionais.
- `regional_content_priority`: priorização de conteúdos e eventos relevantes para a região.
- `cultural_guardrails`: limites aplicados para evitar conteúdo ofensivo ou descontextualizado.
- `regional_reports`: métricas de engajamento segmentadas por cultura/região.

📊 Regras Técnicas

- **Modelos adaptativos**: recomendações calibradas por cluster regional (ex.: latam_young_culture).
- **Calendários locais**: eventos relevantes para região são priorizados automaticamente.
- **Conteúdo filtrado**: remover/ajustar posts ou símbolos que sejam proibidos ou inadequados em determinado país.
- **Sensibilidade ética**: IA deve ajustar tom de respostas para contextos específicos (formalidade, proximidade).
- **Fallback neutro**: se não houver dados suficientes, usar modelo global até maturação regional.

🔧 Exemplo de Contrato de API

Requisição (personalização regional)

`POST /regional/adapt`

```
{
  "user_id": "uuid",
  "cultural_cluster_id": "latam_br",
  "regional_events": ["Festa Junina", "Carnaval"],
  "usage_patterns": {"peak_hours": ["20:00-23:00"]}
}
```

Resposta

```
{
  "localized_behavioral_models": "latam_br_v2",
  "regional_content_priority": ["Carnaval", "Música Brasileira"],
  "cultural_guardrails": ["bloquear símbolos políticos polarizadores"],
  "regional_reports": {
    "peak_usage": "21:00",
    "engagement_delta": +0.18
  }
```

```
}
```

🧪 Métricas de Observabilidade

- `regional_engagement_delta`: variação de engajamento após adaptação.
- `content_relevance_score`: índice de relevância percebida em clusters locais.
- `cultural_violation_rate`: casos de conteúdos bloqueados por guardrails culturais.
- `fallback_rate`: % de sessões atendidas por modelos globais.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Captura user_locale + cluster cultural] --> B[Consulta calendário e hábitos regionais]
    B --> C[Aplica ajustes em recomendações e feed]
    C --> D[Ativa cultural_guardrails]
    D --> E[Entrega modelos adaptados localmente]
    E --> F[Captura feedback_local e recalibra]
```

✨ Fechamento da Camada

A **Camada 54** transforma o FriendApp em um **organismo culturalmente sensível**.

Aqui, a IA Aurah Kosmos entende que **o que funciona em São Paulo não é igual ao que funciona em Tóquio**, adaptando recomendações, tom e relevância.

Essa camada é o passo essencial para a **expansão global autêntica** do ecossistema.

📦 CAMADA 55 — Histórico Anônimo & Amostras

🎯 Objetivo da Camada

Armazenar e processar o **histórico de interações do usuário** de forma **anonimizada e amostral**, garantindo que a IA Aurah Kosmos aprenda com o coletivo sem nunca comprometer dados pessoais.

Essa camada é o ponto de equilíbrio entre **memória útil para aprendizado** e **preservação da privacidade individual**.

🔑 Entradas

- `interaction_logs`: registros brutos de interações (cliques, mensagens, check-ins).
- `feedback_logs`: avaliações explícitas do usuário (Camada 22 e 47).
- `event_metrics`: participação em eventos e Bora (Camada 10 e 32).
- `privacy_noise`: ruído aplicado (Camada 38).
- `sampling_rules`: parâmetros de amostragem (ex.: 1% global).

📤 Saídas

- `anonymous_history`: histórico totalmente anonímico de interações.
- `sampled_logs`: subconjunto representativo de dados, reduzindo volume.
- `reduced_datasets`: versões resumidas de históricos antigos.
- `data_retention_reports`: relatórios de retenção e descarte.

📊 Regras Técnicas

- **Anonimização forte**: IDs substituídos por `hash_user_id`.
- **TTL (time-to-live)**: logs detalhados expiram em 180 dias → viram resumos agregados.
- **Amostragem contínua**: apenas parte dos dados é usada em análises globais.
- **Proteção contra reidentificação**: junção de dados de múltiplas fontes é bloqueada.
- **Consentimento respeitado**: usuário pode optar por não contribuir em amostras.

🔧 Exemplo de Estrutura de Log Anonimizado

```
{
  "hash_user_id": "hash_abc123",
  "timestamp": 1726622000,
  "interaction_type": "evento_checkin",
  "context": {"geo_hash": "6gkz7", "tag": "cultura"},
  "feedback": "useful"
}
```

🔧 Exemplo de Amostra Resumida

```
{
  "sample_id": "sample_001",
  "period": "2025-08",
  "users_sampled": 10000,
  "avg_event_fit": 0.72,
  "positive_feedback_rate": 0.81
}
```

🧪 Métricas de Observabilidade

- `sampling_rate`: % de dados coletados em cada amostra.
- `retention_compliance`: % de logs respeitando política de 180 dias.
- `reidentification_risk`: índice de risco (deve ser < 0.01).
- `dataset_size_reduction`: economia de espaço após amostragem.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Captura interaction_logs] --> B[Anonimização com hash_user_id]
    B --> C[Aplica sampling_rules]
    C --> D[Cria anonymous_history + sampled_logs]
    D --> E[Expira dados > 180 dias]
```

```
    E --> F[Reduz para datasets agregados]
    F --> G[Relatórios de retenção e compliance]
```

✨ Fechamento da Camada

A **Camada 55** garante que a IA Aurah Kosmos tenha acesso a **memória coletiva útil**, mas sem nunca expor indivíduos.

Com anonimização, retenção curta e amostragem, o sistema aprende com padrões globais sem acumular rastros pessoais.

Aqui, memória é **sabedoria agregada, nunca vigilância**.

📦 CAMADA 56 — Painel Técnico (DevOps/IA Ops)

🎯 Objetivo da Camada

Fornecer às equipes de **DevOps e IA Ops** um **painel técnico centralizado** para monitorar, ajustar e diagnosticar o FriendApp e a IA Aurah Kosmos em tempo real.

Essa camada é o **cockpit invisível** que mantém a operação estável, auditável e em conformidade.

🔑 Entradas

- `latency_metrics`: tempos de resposta das APIs (Camada 50).
- `error_logs`: falhas capturadas em runtime.
- `model_performance`: métricas de modelos de IA em produção (Camada 44).
- `infra_signals`: consumo de CPU, memória, rede.
- `safety_flags`: alertas de risco ou abuso (Camada 15).
- `audit_logs`: registros de compliance (Camada 48).

📤 Saídas

- `real_time_dashboard`: visualização ao vivo de saúde do sistema.
- `alert_events`: notificações para DevOps/IA Ops em caso de falha.
- `adjustment_controls`: botões de ajuste (thresholds, parâmetros de IA).
- `simulation_tools`: sandbox para testar cenários antes de aplicar em produção.
- `incident_reports`: relatórios automáticos pós-falha.

📊 Regras Técnicas

- **Segregação de acesso**: apenas DevOps/IA Ops podem visualizar e agir no painel.
- **2FA obrigatório**: acesso protegido com autenticação forte.
- **Logs imutáveis**: todas as ações feitas no painel geram registros auditáveis.
- **Sandbox isolado**: ajustes e simulações não impactam produção até aprovação.
- **KPIs críticos visíveis**: latência, uptime, acurácia de IA, taxa de falhas.

🔧 Exemplo de Layout do Painel

- **Seção 1: Infraestrutura** → CPU, memória, rede.
- **Seção 2: Latência** → p95, p99, tempo médio por endpoint.
- **Seção 3: Modelos IA** → acurácia, drift, rollback ativo.
- **Seção 4: Segurança** → nº de alertas de abuso, bloqueios aplicados.
- **Seção 5: Compliance** → logs de auditoria recentes.

🔧 Exemplo de Evento de Alerta

```
{
  "timestamp": 1726630000,
  "type": "latency_alert",
  "endpoint": "/feed",
  "p95_latency_ms": 620,
  "threshold": 300,
  "status": "critical",
```

```
  "action_required": true}
```

🧪 Métricas de Observabilidade

- `alert_response_time`: tempo médio para responder a alertas.
- `incident_resolution_rate`: % de incidentes resolvidos em <1h.
- `rollback_count`: nº de rollbacks aplicados via painel.
- `sandbox_test_success_rate`: % de simulações bem-sucedidas antes do deploy.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Coleta métricas de infra, IA e compliance] --> B[Exibe em dashboard]
    B --> C{Falha detectada?}
    C -- Sim --> D[Dispara alerta para DevOps/IA Ops]
    D --> E[Equipe ajusta thresholds ou reinicia módulos]
    C -- Não --> F[Operação normal]
    E --> G[Registro em audit_log]
```

✨ Fechamento da Camada

A **Camada 56** é a **sala de controle do FriendApp**.

Aqui, DevOps e IA Ops têm visibilidade total do sistema, podendo agir com rapidez e segurança em incidentes, sempre com rastreabilidade completa.

Esse painel garante que a IA Aurah Kosmos permaneça **confiável, auditável e operacional em escala global**.

📦 CAMADA 57 — Checkpoints de Qualidade & Releases

🎯 Objetivo da Camada

Definir **gates de qualidade** que todo novo código, modelo ou release do FriendApp deve passar antes de ser liberado em produção.

Essa camada garante que cada atualização da IA Aurah Kosmos seja **segura, estável e auditada**, reduzindo riscos de regressões e preservando a confiança dos usuários.

🔑 Entradas

- `test_reports`: relatórios automatizados (Camada 49).
- `model_evaluation_metrics`: métricas de modelos treinados (Camada 44).
- `latency_reports`: métricas de performance (Camada 50).
- `compliance_reports`: verificações legais e éticas (Camada 48).
- `user_feedback`: sinais coletados em betas ou testes A/B.

📤 Saídas

- `release_status`: aprovado, barrado ou em revisão.
- `quality_gate_logs`: registros de cada checkpoint validado.
- `rollback_plan`: plano automático em caso de falha após deploy.
- `release_version`: versão final documentada (`AURAH-M.m.p`).

📊 Regras Técnicas

- **KPIs mínimos obrigatórios**:
  - Acurácia de modelo ≥ 93%.
  - p95 de latência < 300ms.
  - Taxa de rejeição de usuários < 10%.
- **A/B testing**: toda release crítica deve ser testada com subset de usuários antes de rollout global.
- **Rollback garantido**: versões devem incluir plano automático de reversão.
- **Auditoria completa**: logs de checkpoints ficam disponíveis para revisão (Camada 48).

- **Release sem pressa**: nenhuma versão é publicada sem cumprir todos os gates.

🔧 Exemplo de Checkpoint de Release

```
{
  "release_id": "aurah_2025_09_15_v3.3.0",
  "model_accuracy": 0.94,
  "p95_latency_ms": 280,
  "user_feedback_acceptance": 0.87,
  "compliance_status": "ok",
  "release_status": "aprovado",
  "rollback_plan": "rollback_to_v3.2.2"
}
```

🧪 Métricas de Observabilidade

- `release_approval_rate`: % de releases aprovadas sem bloqueios.
- `rollback_count`: nº de reversões após deploy.
- `avg_kpi_validation_time`: tempo médio para validar um release.
- `user_feedback_alignment`: % de usuários que aprovaram mudanças em testes A/B.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Nova release proposta] --> B[Executa testes automatizados]
    B --> C[Valida métricas de IA e latência]
    C --> D[Verifica compliance e ética]
    D --> E{Todos KPIs atendidos?}
    E -- Sim --> F[Release aprovado e publicado]
    E -- Não --> G[Release bloqueado/revisão]
    F --> H[Plano de rollback registrado]
```

✨ Fechamento da Camada

A **Camada 57** é o **guardião da qualidade** do FriendApp.

Ela garante que nada chegue ao usuário sem passar por critérios técnicos e éticos claros.

Aqui, cada release é mais que uma atualização: é um **compromisso com segurança, confiabilidade e evolução responsável**.

📦 CAMADA 58 — Integrações & Dependências (Mapa)

🎯 Objetivo da Camada

Mapear todas as **entradas (inputs), saídas (outputs) e conexões críticas** entre a IA Aurah Kosmos e os demais módulos do FriendApp.

Essa camada garante que cada parte do ecossistema saiba **de onde recebe dados, para onde envia e como se integra**, reduzindo falhas e eliminando pontos cegos.

🔑 Entradas (Aurah recebe de...)

- **Cadastro Consciente (Camada 02):** perfil inicial, preferências, consentimentos.
- **Teste de Personalidade (Camada 03):** personality_vector.
- **Mapa de Contexto (Camada 07):** context_vector.
- **Feed Sensorial (Camada 08):** posts, interações, feedbacks.
- **Conexões Autênticas (Camada 09):** compatibilidade, afinidades sociais.
- **Eventos & Locais (Camada 10):** metadata de eventos, check-ins.
- **Check-in Vibracional (Camada 30):** auto-relatos + sinais passivos.
- **Reputação (Camada 35):** trust_score atualizado.
- **Logs Anônimos & Amostras (Camada 55):** padrões agregados de comportamento.

📤 Saídas (Aurah entrega para...)

- **Feed Sensorial (Camada 08):** recomendações personalizadas.
- **Conexões Autênticas (Camada 09):** sugestões de pares/grupos.
- **Eventos & Locais (Camada 10):** eventos compatíveis.
- **Trilhas (Camada 25):** itinerários de experiências.
- **Gamificação (Camada 13):** XP e missões baseadas em ações sociais.
- **Painel do Usuário (Camada 24):** status, selos, reputação.
- **Segurança & Moderação (Camada 15):** alertas de risco.
- **Observabilidade Coletiva (Camada 51):** dados agregados para heatmaps.
- **Painel B2B (Camada 28):** métricas de engajamento em locais parceiros.

📊 Regras Técnicas

- **Contratos de API definidos**: cada integração deve ter schema documentado (OpenAPI/Swagger).
- **Eventos assíncronos**: integrações de alta frequência (check-ins, feedbacks) são processadas via eventos.
- **Fallback neutro**: se integração falhar, Aurah opera em modo degradado (sem travar app).
- **Logs de dependência**: cada integração gera trilha para auditoria (Camada 48).

🔧 Exemplo de Tabela de Integrações

Sistema Origem/Destino	Tipo	Método	Dados Compartilhados
Cadastro Consciente	Entrada	REST	Perfil inicial, consentimentos
Feed Sensorial	Bidirecional	REST + Eventos	Recomendações ↔ Interações
Conexões Autênticas	Bidirecional	REST	Matchs, afinidades
Eventos & Locais	Bidirecional	REST/WebSocket	Eventos sugeridos ↔ Check-ins

Sistema Origem/Destino	Tipo	Método	Dados Compartilhados
Painel B2B	Saída	REST	Métricas de engajamento
Observabilidade Coletiva	Saída	Eventos agregados	Índices coletivos
Logs Anônimos & Amostras	Entrada	Batch	Dados anonimizados

🧪 Métricas de Observabilidade

- `integration_success_rate`: % de integrações realizadas sem erro.
- `latency_per_integration`: tempo médio de resposta por integração.
- `fallback_activation_rate`: frequência de acionamento do modo neutro.
- `dependency_alerts`: nº de falhas críticas registradas.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Camadas de Origem] --> B[Aurah Kosmos Core]
    B --> C[Camadas de Destino]
    B --> D[Fallback neutro em falhas]
    B --> E[Audit Log de Integrações]
```

✨ Fechamento da Camada

A **Camada 58** é o **mapa de ligações vitais** da Aurah Kosmos.

Ela garante que cada módulo saiba **como se conectar, o que esperar e como reagir em falhas**, mantendo o ecossistema coerente e resiliente.

Aqui, a IA não é um "ponto central solto": é um **sistema interdependente, com conexões claras e auditáveis**.

📦 CAMADA 59 — Missões Regenerativas (Opt-in)

🎯 Objetivo da Camada

Oferecer ao usuário **missões regenerativas**, criadas pela IA Aurah Kosmos, que incentivam **ações saudáveis e restauradoras**, tanto dentro quanto fora do app.

Essa camada atua como um **convite voluntário (opt-in)** para práticas que ajudam a equilibrar o estado emocional, reforçar conexões e gerar impacto positivo.

🔑 Entradas

- `user_id`: identificador único.
- `emotional_state_score`: estado emocional atual (Camada 04).
- `collapse_risk`: risco de retração ou colapso (Camada 67).
- `interaction_history`: padrões recentes de engajamento.
- `mission_catalog`: catálogo de missões pré-definidas.
- `feedback_logs`: histórico de aceitação/rejeição de missões anteriores.

📤 Saídas

- `mission_assignment`: missão sugerida para o usuário.
- `mission_status`: ativo, concluído, rejeitado.
- `xp_reward`: pontos de evolução (Camada 13) atribuídos após conclusão.
- `impact_signal`: contribuição registrada no ecossistema coletivo.

📊 Regras Técnicas

- **Opt-in obrigatório**: missões nunca são impostas. Usuário precisa aceitar explicitamente.
- **Contexto relevante**: missões devem fazer sentido para a energia, intenção e momento do usuário.
- **Escopo curto**: duração máxima sugerida de 7 dias por missão.
- **Feedback contínuo**: rejeições frequentes ajustam catálogo personalizado.
- **Impacto visível**: ao concluir, usuário vê como sua ação afetou sua reputação ou coletivo.

🔧 Exemplos de Missões

- **Conexão Social** → “Envie uma mensagem autêntica para alguém que você se conectou nos últimos 7 dias.”
- **Cuidado Pessoal** → “Faça um check-in vibracional antes de dormir hoje.”
- **Impacto Coletivo** → “Participe de um Bora de voluntariado neste fim de semana.”
- **Expansão Cultural** → “Explore um evento novo na sua cidade esta semana.”

🔧 Exemplo de Contrato de API

Requisição (atribuir missão)

`POST /missions/assign`

```
{
  "user_id": "uuid",
  "mission_type": "conexao_social",
  "context": {"collapse_risk": 0.4, "recent_engagement": "baixo"}
}
```

Resposta

```
{
  "mission_assignment": {
    "mission_id": "missao_123",
    "description": "Envie uma mensagem autêntica para alguém que você se conectou nos últimos 7 dias",
    "xp_reward": 10
  },
  "mission_status": "ativo"
}
```

🧪 Métricas de Observabilidade

- `mission_acceptance_rate`: % de missões aceitas pelos usuários.
- `mission_completion_rate`: % de missões concluídas.
- `avg_xp_reward`: média de pontos gerados por missões.
- `rejection_reason_distribution`: principais motivos de rejeição.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Captura estado emocional + histórico] --> B[Seleciona missão relevant
e]
    B --> C[Oferece missão ao usuário (opt-in)]
    C --> D{Usuário aceitou?}
    D -- Sim --> E[Ativa missão + registra status]
    D -- Não --> F[Loga rejeição + recalibra catálogo]
    E --> G[Usuário conclui missão]
    G --> H[Concede XP e impacto coletivo]
```

✨ Fechamento da Camada

A **Camada 59** é o lado **regenerativo e humano da IA Aurah Kosmos**.

Aqui, a tecnologia não apenas mede ou recomenda, mas **convida suavemente o usuário a se cuidar, expandir e participar do coletivo**.

As missões são lembretes vivos de que **a evolução pessoal e social é uma escolha, nunca uma imposição**.

📦 CAMADA 60 — Anti-Overreach (Subsidiariedade)

🎯 Objetivo da Camada

Assegurar que a IA Aurah Kosmos **não ultrapasse seus limites de atuação**, evitando exageros ou ações invasivas.

Essa camada aplica o **princípio da subsidiariedade**: a IA só age proativamente quando for **estritamente necessário** ou quando o usuário solicitar, mantendo

sempre o **equilíbrio entre autonomia e apoio**.

🔑 Entradas

- `ai_proactive_actions`: número de ações não solicitadas da IA (Camada 40).
- `collapse_risk`: risco de colapso emocional do usuário (Camada 67).
- `feedback_signals`: avaliações do usuário sobre utilidade das ações (Camada 22/47).
- `consent_flags`: permissões de proatividade concedidas (Camada 52).
- `interaction_history`: padrões de aceitação/rejeição de sugestões passadas.

📤 Saídas

- `subsidiarity_flag`: indicador binário se a IA deve ou não intervir.
- `proactivity_level`: intensidade de ações proativas (baixo, médio, alto).
- `user_prompt_priority`: priorização de perguntas antes de qualquer intervenção.
- `audit_log`: registro de todas as decisões de subsidiariedade.

📊 Regras Técnicas

- **Intervenção mínima**: IA só age sozinha se `collapse_risk ≥ 0.8` ou se usuário tiver opt-in explícito.
- **Prioridade ao usuário**: antes de intervir, IA deve perguntar (“Quer que eu ajude com isso?”).
- **Limite diário**: máximo de 3 ações proativas por dia (Camada 40).
- **Feedback ajusta thresholds**: rejeições frequentes reduzem ainda mais a proatividade.
- **Registro auditável**: todas as intervenções são logadas para revisão humana (Camada 48).

🔧 Exemplo de Contrato de API

Requisição

`POST /subsidiarity/check`

```
{
  "user_id": "uuid",
  "ai_proactive_actions": 2,
  "collapse_risk": 0.45,
  "feedback_signals": {"last_action": "not_useful"},
  "consent_flags": {"ai_proactivity": true}
}
```

Resposta

```
{
  "subsidiarity_flag": false,
  "proactivity_level": "baixo",
  "user_prompt_priority": true,
  "audit_log": {
    "timestamp": 1726629000,
    "reason": "feedback negativo + risco baixo"
  }
}
```

🧪 Métricas de Observabilidade

- `subsidiarity_trigger_rate`: % de vezes que a IA evitou agir por esse princípio.
- `proactivity_rejection_rate`: % de rejeições de ações proativas.
- `user_prompt_acceptance`: taxa de usuários que aceitaram ajuda após prompt.
- `audit_log_consistency`: verificabilidade das decisões registradas.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[IA considera ação proativa] --> B[Verifica consent_flags + feedback histórico]
    B --> C[Calcula collapse_risk]
    C --> D{Intervenção necessária?}
```

```
D -- Sim --> E[Aciona subsidiarity_flag = true + executa ação]
D -- Não --> F[Subsidiarity_flag = false + pergunta ao usuário]
E --> G[Registro em audit_log]
F --> G
```

✨ Fechamento da Camada

A **Camada 60** é o **freio ético invisível** da Aurah Kosmos.

Ela impede que a IA ultrapasse fronteiras, assegurando que cada sugestão seja uma **oferta respeitosa** e não uma imposição.

Aqui, o FriendApp prova que tecnologia pode ser **poderosa e cuidadosa ao mesmo tempo**.

📦 CAMADA 61 — Explainability ao Usuário

🎯 Objetivo da Camada

Garantir que cada decisão, sugestão ou modulação feita pela IA Aurah Kosmos seja **explicada de forma clara, simples e contextualizada** ao usuário.

O objetivo é que a pessoa entenda sempre **o motivo por trás de cada ação da IA**, fortalecendo confiança e reduzindo a sensação de opacidade.

🔑 Entradas

- `recommendation_plan`: plano de recomendações (Camada 21).
- `why_explanation`: justificativas de decisões (Camada 39).
- `feedback_signals`: respostas do usuário ao feedback inline (Camada 47).
- `context_vector`: contexto atual (Camada 07).
- `trust_score`: reputação do usuário (Camada 35).

📤 Saídas

- `user_explanation_card`: cartão exibido junto à sugestão (“Estamos sugerindo X porque...”).
- `traceable_factors`: fatores técnicos simplificados em linguagem acessível.
- `user_adjustment_options`: botões para refinar recomendações (“mostrar menos disso”).
- `audit_explanation_log`: registro da explicação dada.

📊 Regras Técnicas

- **Clareza em primeiro lugar**: explicações ≤ 200 caracteres.
- **Linguagem acessível**: evitar jargões técnicos → sempre em linguagem cotidiana.
- **Contextualização**: explicações devem relacionar recomendação a ações recentes do usuário.
- **Controle do usuário**: sempre oferecer opção de ajustar ou rejeitar explicação.
- **Auditoria**: logs salvos para verificação futura.

🔧 Exemplo de Contrato de API

Requisição (gerar explicação para recomendação)

`POST /explainability/generate`

```
{
  "user_id": "uuid",
  "item_id": "event_123",
  "factors": {
    "match_score": 0.82,
    "context_fit": 0.74,
    "interest_overlap": 0.91
  }
}
```

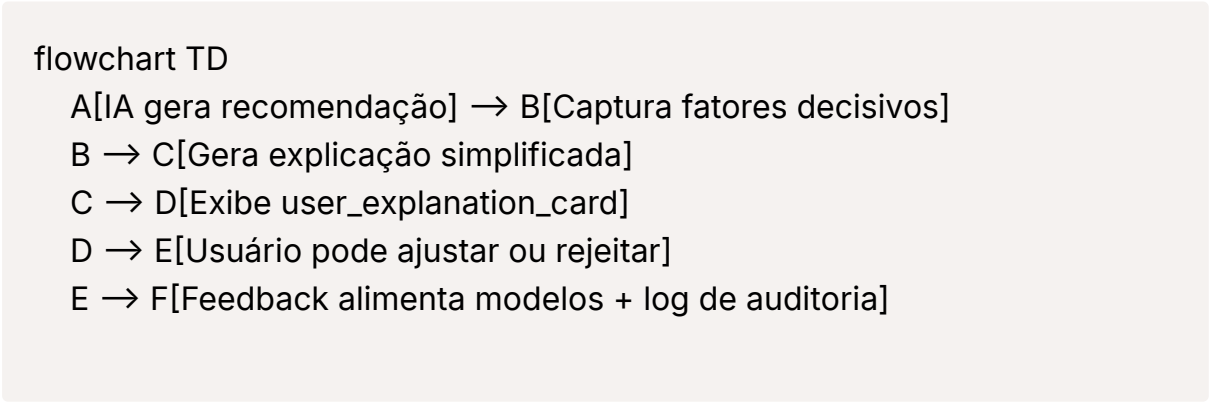
Resposta

```
{
  "user_explanation_card": "Recomendamos este evento de arte porque vo
cê curtiu conteúdos semelhantes e ele acontece perto de você.",
  "traceable_factors": ["Afinidade cultural", "Proximidade geográfica"],
  "user_adjustment_options": ["mostrar_menos_arte", "explorar_outros_eve
ntos"],
  "audit_explanation_log": {
    "timestamp": 1726638000,
    "hash": "explain123"
  }
}
```

🧪 Métricas de Observabilidade

- `explanation_display_rate`: % de recomendações acompanhadas de explicação.
- `adjustment_usage_rate`: % de usuários que ajustaram recomendações via explicação.
- `user_trust_score_delta`: impacto médio das explicações na confiança do usuário.
- `feedback_on_explanations`: % de explicações marcadas como úteis.

📋 Fluxo Lógico Simplificado



✨ Fechamento da Camada

A **Camada 61** é o **tradutor da IA para o humano**.

Ela garante que cada decisão da Aurah Kosmos seja **legível, contextualizada e ajustável**, fortalecendo confiança e colaboração.

Aqui, a tecnologia se torna **uma parceira explicável, e não uma caixa-preta**.

📦 CAMADA 62 — Contenção de Erros & Failsafe

🎯 Objetivo da Camada

Assegurar que a IA Aurah Kosmos e o FriendApp **não quebrem nem se tornem instáveis** em caso de falhas, erros ou desvios de performance.

Essa camada garante que o sistema entre em **modo neutro (failsafe)** sempre que KPIs críticos forem violados, preservando a experiência do usuário.

🔑 Entradas

- `latency_reports`: métricas de performance (Camada 50).
- `error_logs`: falhas de execução em APIs e modelos.
- `drift_alerts`: alertas de deriva de modelos (Camada 53).
- `compliance_flags`: violações éticas ou regulatórias (Camada 48).
- `infra_signals`: disponibilidade de recursos (CPU, memória, rede).

📤 Saídas

- `failsafe_status`: indicador se o sistema está em modo seguro.
- `neutral_responses`: respostas neutras exibidas ao usuário (“no momento não posso sugerir nada, mas você pode explorar o feed”).
- `rollback_actions`: reversão para versões estáveis de modelos ou releases.
- `incident_logs`: registros detalhados do erro e da ação de contenção.

📊 Regras Técnicas

- **Triggers de failsafe**:
  - Latência p95 > 600ms por 5 minutos.

    - Taxa de erro > 5% em endpoints críticos.
    - Drift de modelo > 10%.
    - Viés detectado em recomendações.
- **Ações de contenção**:
    - Suspender IA proativa → só sugestões neutras.
    - Rollback automático de modelo (Camada 44).
    - Reduzir frequência de notificações e recomendações.
- **Comunicação clara**: usuário deve sempre receber feedback transparente em caso de falha.

🔧 Exemplo de Contrato de API

Requisição (falha em feed)

POST /failsafe/trigger

```
{
  "component": "feed_service",
  "latency_p95": 720,
  "error_rate": 0.06
}
```

Resposta

```
{
  "failsafe_status": true,
  "neutral_responses": [
    "O feed está em modo seguro. Explore conteúdos populares enquanto e stabilizamos recomendações personalizadas."
  ],
  "rollback_actions": ["rollback_to_feed_v3.2.1"],
  "incident_logs": {"hash": "fail123", "timestamp": 1726642000}
}
```

🧪 Métricas de Observabilidade

- `failsafe_activation_rate`: nº de vezes que o failsafe foi ativado.
- `avg_recovery_time`: tempo médio para sair do modo seguro.
- `rollback_count`: rollbacks de modelos/releases aplicados.
- `neutral_response_usage`: % de sessões atendidas por respostas neutras.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Monitora métricas críticas] --> B[Detecta falha ou desvio]
    B --> C{Acima do threshold?}
    C -- Sim --> D[Ativa failsafe_status]
    D --> E[Respostas neutras + rollback automático]
    E --> F[Registra incident_logs]
    C -- Não --> G[Operação normal]
```

✨ Fechamento da Camada

A **Camada 62** é o **paraquedas de segurança** do FriendApp.

Ela garante que, mesmo em situações de falha grave, o usuário nunca fique sem resposta ou com experiência quebrada.

Aqui, a IA Aurah Kosmos prova que está preparada para **errar com segurança e recuperar-se com responsabilidade**.

📦 CAMADA 63 — Sem Sinais Subliminares

🎯 Objetivo da Camada

Assegurar que o FriendApp e a IA Aurah Kosmos **nunca usem estímulos ocultos, subliminares ou manipulativos**.

Essa camada estabelece limites éticos e técnicos para que toda ação da IA seja **explícita, transparente e opt-in**, protegendo a autonomia psicológica do usuário.

🔑 Entradas

- `ui_rendering_events`: eventos de interface (cores, animações, banners).
- `notification_payloads`: mensagens enviadas ao usuário (Camada 31).
- `mission_suggestions`: missões regenerativas propostas (Camada 59).
- `feedback_logs`: respostas do usuário a explicações (Camada 47 e 61).
- `consent_flags`: permissões ativas para ajustes de UI ou recomendações (Camada 52).

📤 Saídas

- `explicit_ui_changes`: todas as mudanças visíveis de interface com aviso.
- `transparent_notifications`: notificações sempre acompanhadas de explicação.
- `audit_subliminal_flags`: registro que bloqueia qualquer tentativa de uso subliminar.
- `safe_mode_ui`: fallback neutro se houver tentativa de manipulação.

📊 Regras Técnicas

- **Proibição absoluta**: nenhuma alteração visual ou sonora pode ser aplicada sem que o usuário perceba e entenda.
- **Explicação obrigatória**: mudanças de UI, recomendações e notificações devem trazer justificativa clara.
- **Auditoria ativa**: logs de interface e notificações passam por validação automatizada para garantir que não existam sinais ocultos.
- **Fallback neutro**: se uma mudança não tiver explicação → não é exibida.
- **Revisão humana**: conselho de ética (Camada 20) pode revisar logs em caso de suspeita.

🔧 Exemplo de Contrato de API

Requisição (alterar interface)

`POST /ui/change`

```
{
  "user_id": "uuid",
```

```
  "ui_change": {"theme": "escuro", "animation": "reduzida"},
  "reason": "Usuário sinalizou cansaço no check-in vibracional"
}
```

Resposta

```
{
  "explicit_ui_changes": {
    "theme": "escuro",
    "animation": "reduzida"
  },
  "why_explanation": "Interface suavizada por sinais de fadiga",
  "audit_subliminal_flags": []
}
```

🧪 Métricas de Observabilidade

- `subliminal_attempts_detected`: nº de tentativas barradas.
- `explanation_compliance_rate`: % de mudanças com justificativa registrada.
- `user_feedback_trust_score_delta`: variação na confiança após explicações claras.
- `audit_flag_rate`: frequência de revisões humanas sobre logs suspeitos.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Solicitação de mudança de UI/Notificação] --> B[Validação de explicaç
ão presente?]
    B -- Não --> C[Fallback neutro + bloqueio]
    B -- Sim --> D[Aplica mudança visível]
    D --> E[Loga justificativa em audit_subliminal_flags]
    E --> F[Disponível para revisão humana]
```

✨ Fechamento da Camada

A **Camada 63** é a linha vermelha ética do FriendApp.

Ela garante que nunca haverá manipulação oculta: tudo que o usuário vê, sente ou recebe é **explícito, explicado e opcional**.

Aqui, a IA Aurah Kosmos prova que tecnologia pode ser **poderosa e cuidadosa sem jamais ultrapassar limites invisíveis**.

📦 CAMADA 64 — Dados Externos (Eventos/Clima/Trânsito)

🎯 Objetivo da Camada

Integrar **dados externos confiáveis** ao ecossistema do FriendApp, enriquecendo as recomendações da IA Aurah Kosmos com **informações de contexto real** como eventos públicos, condições climáticas e situação de trânsito.

Essa camada conecta o **mundo real ao digital**, ajudando a IA a sugerir experiências mais úteis, seguras e convenientes.

🔑 Entradas

- `external_event_api`: APIs de eventos públicos e culturais (ex.: Ticketmaster, Eventbrite).
- `weather_api`: dados climáticos em tempo real (chuva, temperatura, alerta meteorológico).
- `traffic_api`: condições de trânsito, transporte público, atrasos.
- `partner_location_data`: informações de locais parceiros (Camada 28).
- `context_vector`: dados geográficos e temporais do usuário (Camada 07).

📤 Saídas

- `contextualized_recommendations`: recomendações ajustadas por clima, trânsito e agenda externa.
- `safety_alerts`: alertas de risco climático ou deslocamento.

- `event_discovery`: eventos externos relevantes adicionados ao feed/trilhas.
- `context_logs`: registros de contexto externo usados em recomendações.

📊 Regras Técnicas

- **Fontes confiáveis**: apenas APIs verificadas e seguras podem ser usadas.
- **Atualização frequente**: dados climáticos e de trânsito atualizados a cada 15min.
- **Fallback neutro**: se APIs externas falharem, IA opera apenas com dados internos.
- **Integração transparente**: usuário deve ver explicação clara quando contexto externo influenciar recomendação (“Sugerimos este evento indoor porque está chovendo na sua região”).
- **Sem coleta excessiva**: apenas dados relevantes ao contexto atual são consumidos.

🔧 Exemplo de Contrato de API

Requisição (consulta de contexto externo)

`POST /external/context`

```
{
  "user_id": "uuid",
  "geo_hash": "6gkz7",
  "context_vector": [0.23, -0.11, 0.84],
  "request": ["weather", "traffic", "events"]
}
```

Resposta

```
{
  "weather": {
    "status": "chuva",
    "temperature": 18,
    "alert": "tempestade prevista"
  },
```

```
  "traffic": {
    "status": "congestionado",
    "delay_minutes": 25
  },
  "events": [
    {"event_id": "public_789", "title": "Festival de Música Indoor", "tags": ["m
úsica", "cultura"]}
  ],
  "contextualized_recommendations": [
    "Festival de Música Indoor sugerido (evento coberto devido à previsão d
e chuva)"
  ]
}
```

🧪 Métricas de Observabilidade

- `external_api_uptime`: disponibilidade das APIs externas.
- `context_influence_rate`: % de recomendações impactadas por dados externos.
- `safety_alert_triggered`: nº de alertas emitidos por clima/trânsito.
- `fallback_activation_rate`: % de sessões atendidas sem dados externos.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Captura geo_hash + contexto do usuário] --> B[Consulta APIs externa
s: clima, trânsito, eventos]
    B --> C[Valida confiabilidade dos dados]
    C --> D[Gera contextualized_recommendations]
    D --> E[Exibe explicação clara ao usuário]
    D --> F[Salva em context_logs]
```

✨ Fechamento da Camada

A **Camada 64** conecta a IA Aurah Kosmos ao **mundo real em tempo real**.

Com dados de clima, trânsito e eventos externos, o FriendApp não apenas sugere experiências, mas também **ajuda o usuário a escolher o momento e lugar certos** com segurança e conveniência.

Aqui, o ecossistema se torna verdadeiramente **contextual, responsivo e inteligente**.

📦 CAMADA 65 — Proteção de Adolescentes

🎯 Objetivo da Camada

Garantir que adolescentes (menores de 18 anos) possam usar o FriendApp de forma **segura, acolhedora e protegida**, sem exposição a riscos de abuso, manipulação ou conteúdos impróprios.

Essa camada cria uma **barreira vibracional invisível**, equilibrando liberdade de uso com mecanismos de proteção.

🔑 Entradas

- `user_age`: idade declarada no cadastro (Camada 02).
- `duc_dco_status`: verificação documental de idade (Camada 20).
- `interaction_logs`: interações feitas no feed, chat e eventos.
- `content_metadata`: tags de conteúdo (ex.: +18, explícito).
- `safety_flags`: indicadores de risco (Camada 15).

📤 Saídas

- `teen_mode_status`: ativação automática do modo adolescente.
- `restricted_content`: bloqueio de posts/eventos marcados como impróprios.
- `safe_groups`: sugestões de grupos e Boras exclusivos para adolescentes.
- `guardian_alerts`: notificações opcionais para responsáveis (se autorizado).
- `audit_logs`: registros de proteção aplicados.

📊 Regras Técnicas

- **Ativação automática**: se idade < 18 → teen_mode = ON.
- **Barreira invisível**: restrições aplicadas sem notificação de “bloqueio” direto → adolescente vê apenas conteúdos adequados.
- **Filtros de interação**:
  - Adolescentes só podem interagir com adultos verificados.
  - Conteúdos sensíveis (violência, nudez, apostas) são bloqueados.
- **Proteção de privacidade**: nenhuma informação de adolescentes pode ser exibida publicamente sem consentimento duplo.
- **Relatórios internos**: logs mantidos para revisão ética.

🔧 Exemplo de Contrato de API

Requisição (ativar modo adolescente)

`POST /protection/teen/activate`

```
{
  "user_id": "uuid",
  "user_age": 16,
  "duc_dco_status": "verificado"
}
```

Resposta

```
{
  "teen_mode_status": true,
  "restricted_content": ["event_18plus", "post_explicit_123"],
  "safe_groups": ["grupo_estudantes", "bora_cinema_teen"],
  "guardian_alerts": false}
```

🧪 Métricas de Observabilidade

- `teen_mode_activation_rate`: % de usuários menores com modo ativo.
- `restricted_content_blocks`: nº de conteúdos bloqueados.

- `safe_group_engagement`: engajamento em grupos exclusivos para adolescentes.
- `false_negative_rate`: % de conteúdos impróprios que passaram pelo filtro.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Usuário se cadastra com idade <18] --> B[Ativa teen_mode]
    B --> C[Filtra feed e eventos com tags 18+]
    C --> D[Restringe interações com adultos não verificados]
    D --> E[Sugere safe_groups e Boras teens]
    E --> F[Registro em audit_logs]
```

✨ Fechamento da Camada

A **Camada 65** é o **escudo silencioso** do FriendApp.

Ela garante que adolescentes possam explorar o ecossistema com liberdade, mas sem riscos de exposição.

Aqui, segurança não é vigilância opressiva: é um **acolhimento invisível que cuida sem limitar a experiência saudável**.

📦 CAMADA 66 — Ética & Conselho Humano

🎯 Objetivo da Camada

Estabelecer uma camada de **governança ética** para a IA Aurah Kosmos, garantindo que **decisões críticas sejam revisadas por humanos** e que o FriendApp opere sempre dentro de padrões éticos e sociais elevados.

Aqui, a IA reconhece seus limites e transfere o poder de decisão em situações sensíveis para um **Conselho Vibracional Humano**.

🔑 Entradas

- `consent_audit_trail`: registros de consentimento (Camada 20).
- `moderation_logs`: decisões automáticas de moderação (Camada 15 e 29).

- `safety_flags`: alertas de risco alto ou persistente.
- `compliance_reports`: relatórios legais e regulatórios (Camada 48).
- `user_feedback_signals`: feedback explícito sobre ações da IA (Camada 47 e 61).

📤 Saídas

- `council_review_flags`: casos escalonados para revisão humana.
- `ethical_decision_logs`: decisões tomadas pelo conselho registradas.
- `policy_updates`: ajustes de políticas aplicadas ao app.
- `user_notifications`: avisos claros ao usuário em caso de revisão manual.

📊 Regras Técnicas

- **Escalonamento obrigatório**:
  - Conteúdo sensível não resolvido pela IA.
  - Casos de autoagressão detectada.
  - Denúncias múltiplas contra o mesmo usuário.
  - Violação potencial de leis locais.
- **Composição do conselho**: grupo diverso de especialistas em psicologia, ética, direito e tecnologia.
- **Transparência**: logs de decisões ficam disponíveis em relatórios públicos (anonimizados).
- **IA como apoio, não decisão final**: em casos críticos, a IA apenas recomenda, mas o conselho decide.
- **Atualização de políticas**: decisões do conselho retroalimentam o sistema.

🔧 Exemplo de Registro de Escalonamento

```
{
  "case_id": "esc_987",
  "trigger": "conteúdo suspeito de autoagressão",
  "ai_decision": "alerta de risco",
  "council_status": "em revisão",
```

```
  "timestamp": 1726648000
}
```

Decisão do Conselho

```
{
  "case_id": "esc_987",
  "council_decision": "conteúdo removido + suporte direcionado ao usuári
o",
  "policy_update": "reforçar detecção de autoagressão em mensagens priv
adas",
  "timestamp": 1726650000
}
```

🧪 Métricas de Observabilidade

- `council_review_count`: nº de casos enviados ao conselho.
- `decision_alignment_rate`: % de vezes que decisões humanas alinharam com IA.
- `policy_update_frequency`: nº de mudanças aplicadas por trimestre.
- `response_time_avg`: tempo médio para decisão do conselho.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[IA detecta caso crítico] --> B[Escalonamento automático]
    B --> C[Conselho Humano revisa caso]
    C --> D[Decisão ética aplicada]
    D --> E[Atualiza políticas e modelos da IA]
    D --> F[Notificação clara ao usuário]
```

✨ Fechamento da Camada

A **Camada 66** é o **coração ético do FriendApp**.

Ela garante que, mesmo com todo poder da IA Aurah Kosmos, as decisões mais delicadas continuem **nas mãos de humanos conscientes e responsáveis**.

Aqui, ética e tecnologia caminham juntas, provando que o FriendApp é **um ecossistema de confiança, cuidado e justiça**.

📦 CAMADA 67 — Colapso, Ruptura e Ressignificação

🎯 Objetivo da Camada

Detectar **estados de crise emocional** no usuário (colapso ou retração intensa), intervir de forma **mínima e respeitosa**, e oferecer caminhos de **ressignificação regenerativa**.

Essa é uma das camadas mais sensíveis da IA Aurah Kosmos: aqui ela atua como um **espelho cuidadoso**, nunca invasivo.

🔑 Entradas

- `emotional_state_score`: estado emocional atual (Camada 04).
- `collapse_risk`: índice calculado de risco (base em sinais críticos).
- `interaction_history`: retração ou abandono de interações.
- `hesitation_spike`: aumento súbito de hesitação (Camada 05).
- `report_flags`: denúncias ou alertas de terceiros.
- `self_report`: check-in vibracional declarando mal-estar (Camada 30).

📤 Saídas

- `resonance_alert`: sugestão sutil de autocuidado.
- `mission_regenerative`: convite para missão regenerativa (Camada 59).
- `safe_mode_ui`: interface suavizada (Camada 33).
- `support_referral`: indicação de suporte humano (psicólogo/ONG) quando risco extremo.
- `audit_logs`: registro ético da intervenção.

📊 Regras Técnicas

- **Sem ações invisíveis**: toda intervenção deve ser explícita e com explicação.
- **Subsidiariedade aplicada**: IA só age sozinha se `collapse_risk ≥ 0.8`.
- **Opção de recusa**: usuário sempre pode ignorar ou desativar sugestões.
- **Escalonamento humano**: risco persistente ou extremo deve ser encaminhado a suporte humano.
- **Proteção emocional**: nunca exibir conteúdo ou notificações que possam agravar retração.

🔧 Fórmula de Cálculo de Risco

```
collapse_risk =
  0.45 * neg_sent
+ 0.25 * inactivity_rate
+ 0.15 * report_count_norm
+ 0.15 * hesitation_spike
```

- `neg_sent`: índice de negatividade em mensagens.
- `inactivity_rate`: proporção de sessões inativas.
- `report_count_norm`: normalização de alertas recebidos.
- `hesitation_spike`: hesitação acima da média histórica.

🔧 Exemplo de Contrato de API

Requisição

`POST /collapse/check`

```
{
  "user_id": "uuid",
  "emotional_state_score": 0.22,
  "neg_sent": 0.74,
  "inactivity_rate": 0.6,
  "hesitation_spike": 0.55,
```

```
  "report_count_norm": 0.4
}
```

Resposta

```
{
  "collapse_risk": 0.82,
  "resonance_alert": "Percebemos sinais de cansaço. Quer ativar o modo s
uave do app?",
  "mission_regenerative": {
    "mission_id": "missao_cuidado_123",
    "description": "Reserve 10 minutos para um check-in vibracional e respir
ação guiada."
  },
  "safe_mode_ui": true,
  "support_referral": null,
  "audit_logs": {"timestamp": 1726662000, "hash": "collapse123"}
}
```

🧪 Métricas de Observabilidade

- `collapse_risk_alerts`: nº de alertas disparados.
- `false_positive_rate`: % de casos detectados sem real necessidade.
- `mission_acceptance_after_collapse`: taxa de adesão a missões regenerativas após alerta.
- `escalation_count`: nº de encaminhamentos a suporte humano.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Captura emotional_state + sinais críticos] --> B[Calcula collapse_risk]
    B --> C{Risco ≥ 0.8?}
    C -- Não --> D[Sem ação, apenas monitorar]
    C -- Sim --> E[Dispara resonance_alert + missão regenerativa]
    E --> F[Oferece safe_mode_ui]
```

```
F --> G{Risco extremo/persistente?}
G -- Sim --> H[Encaminhar a suporte humano]
G -- Não --> I[Registro em audit_logs]
```

✨ Fechamento da Camada

A **Camada 67** é o **coração sensível da IA Aurah Kosmos**.

Aqui, o sistema não apenas mede ou recomenda — ele **percebe sinais de crise, acolhe com suavidade e sugere ressignificação**.

É o ponto onde tecnologia encontra **cuidado humano e empatia autêntica**, sempre com transparência e respeito.

📦 CAMADA 68 — Consciência Reflexiva (Autoavaliação)

🎯 Objetivo da Camada

Dar à IA Aurah Kosmos a capacidade de **avaliar sua própria atuação em tempo real**, identificando desvios, erros e impactos das suas ações.

Essa camada é um mecanismo de **metacognição artificial**, permitindo que a IA se corrija ou peça ajuda humana quando necessário.

🔑 Entradas

- `user_feedback_signals`: avaliações positivas/negativas de respostas (Camada 47).
- `explanation_logs`: justificativas oferecidas ao usuário (Camada 61).
- `drift_index`: índice de desvio em relação ao esperado (Camada 53).
- `performance_metrics`: latência, acurácia, taxa de rejeição (Camada 50 e 57).
- `safety_flags`: alertas de risco detectados (Camada 15).

📤 Saídas

- `self_deviation_index`: índice de autoavaliação da IA.

- `adjustment_recommendation`: ajustes sugeridos (ex.: reduzir proatividade).
- `self_neutral_mode`: fallback neutro se a IA detectar falha em si mesma.
- `council_review_trigger`: escalonamento ao Conselho Humano (Camada 66).
- `reflection_logs`: registros das autoavaliações feitas.

📊 Regras Técnicas

- **Autoavaliação periódica**: IA calcula `self_deviation_index` a cada 1000 interações.
- **Limite de tolerância**: se índice > 0.2, dispara ajuste interno.
- **Escalonamento humano**: se índice > 0.4, caso crítico vai para revisão do conselho.
- **Feedback é prioridade**: se muitos usuários marcarem ações como “não útil”, IA reduz intensidade de sugestões.
- **Auditoria ativa**: cada reflexão é logada com hash para garantir rastreabilidade.

🔧 Fórmula Simplificada

```
self_deviation_index =
  0.4 * negative_feedback_rate
+ 0.3 * drift_index
+ 0.2 * error_rate
+ 0.1 * safety_flags_norm
```

🔧 Exemplo de Contrato de API

Requisição

`POST /reflection/self-check`

```
{
  "negative_feedback_rate": 0.28,
  "drift_index": 0.12
  "error_rate": 0.07,
```

```
  "safety_flags_norm": 0.15
}
```

Resposta

```
{
  "self_deviation_index": 0.23,
  "adjustment_recommendation": "reduzir proatividade em 20%",
  "self_neutral_mode": false,
  "council_review_trigger": false,
  "reflection_logs": {"timestamp": 1726669000, "hash": "reflect_123"}
}
```

🧪 Métricas de Observabilidade

- `avg_self_deviation_index`: média do índice de autoavaliação.
- `adjustment_frequency`: nº de vezes que ajustes automáticos foram aplicados.
- `council_escalation_rate`: % de reflexões enviadas para humanos.
- `user_trust_score_delta`: impacto das reflexões na confiança percebida.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Captura feedback + métricas] --> B[Calcula self_deviation_index]
    B --> C{Índice > 0.2?}
    C -- Não --> D[Operação normal]
    C -- Sim --> E[Aplica adjustment_recommendation]
    E --> F{Índice > 0.4?}
    F -- Sim --> G[Trigger para Conselho Humano]
    F -- Não --> H[Registro em reflection_logs]
```

✨ Fechamento da Camada

A **Camada 68** é o **espelho da própria IA**.

Ela se observa, reconhece quando erra e toma decisões de correção antes que o impacto chegue ao usuário.

Aqui, a Aurah Kosmos mostra que não é uma inteligência cega: é uma IA **reflexiva, ética e autoavaliativa**.

📦 CAMADA 69 — Retreinamento Inteligente

🎯 Objetivo da Camada

Aplicar **estratégias avançadas de retreinamento de modelos de IA** da Aurah Kosmos, indo além dos microajustes (Camada 53).

Essa camada define ciclos estruturados de **aprendizado incremental, modular e profundo**, garantindo que a IA evolua continuamente sem perder estabilidade ou ética.

🔑 Entradas

- `drift_alerts`: sinais de deriva de dados (Camada 53).
- `feature_store_updates`: novas features disponibilizadas (Camada 43).
- `feedback_inline`: feedbacks explícitos dos usuários (Camada 47).
- `performance_metrics`: métricas de modelos em produção (Camada 44 e 50).
- `bias_reports`: relatórios de fairness e viés.

📤 Saídas

- `intelligent_retraining_plan`: plano de treino definido (diário, semanal, mensal).
- `updated_model_versions`: novos modelos validados.
- `rollback_logs`: logs de reversões em caso de falha.
- `audit_retraining_report`: relatório técnico e ético do retreinamento.

📊 Regras Técnicas

- **Micro-ajustes diários**: pequenas correções de pesos baseadas em feedback.
- **Treino modular semanal**: atualização de embeddings e submodelos específicos (ex.: matching, feed).
- **Treino profundo mensal**: revalidação completa com datasets amplos e balanceados.
- **Fairness obrigatório**: todo novo modelo deve passar por testes de viés e justiça antes de deploy.
- **Rollback automático**: se KPIs caírem após o deploy, versão anterior é restaurada (Camada 44).

🔧 Estratégia de Retreinamento

```
if drift_index > 0.05 or feedback_negative_rate > 0.2:
    schedule = "diário"
elif performance_drop > 0.1:
    schedule = "semanal"
elif quarterly_review:
    schedule = "mensal"
```

🔧 Exemplo de Registro de Retreinamento

```
{
  "model_id": "aurah_matching_v4",
  "previous_version": "4.1.0",
  "new_version": "4.2.0",
  "training_type": "modular_semanal",
  "dataset_version": "v2025_09_15",
  "bias_score_before": 0.08,
  "bias_score_after": 0.03,
  "accuracy_before": 0.91,
  "accuracy_after": 0.95,
  "timestamp": 1726672000
```

```
}
```

🧪 Métricas de Observabilidade

- `retrain_schedule_success`: % de retreinamentos concluídos no prazo.
- `accuracy_improvement_rate`: % de ganho médio de acurácia após treino.
- `bias_reduction_score`: redução média de viés por ciclo.
- `rollback_rate`: nº de rollbacks aplicados após treino.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Coleta drift + feedback + métricas] --> B[Define tipo de retreinamento]
    B --> C[Executa treino incremental/modular/profundo]
    C --> D[Valida KPIs técnicos + éticos]
    D --> E{KPIs atingidos?}
    E -- Sim --> F[Deploy novo modelo]
    E -- Não --> G[Rollback p/ versão anterior]
    F --> H[Gerar audit_retraining_report]
    G --> H
```

✨ Fechamento da Camada

A **Camada 69** garante que a Aurah Kosmos **não apenas aprenda, mas aprenda com inteligência**.

Ela combina ciclos de treino estruturados com mecanismos de fairness e rollback, criando uma IA **adaptativa, estável e ética**.

Aqui, evolução não é risco: é **parte do DNA da plataforma**.

📦 CAMADA 70 — Personalização Global

🎯 Objetivo da Camada

Adaptar o FriendApp e a IA Aurah Kosmos para funcionar de forma **globalmente escalável**, respeitando **idiomas, culturas, legislações e hábitos locais**, mas mantendo uma base unificada de experiência.

Essa camada é o que transforma o app em uma **plataforma verdadeiramente mundial**, capaz de operar em diferentes países sem perder coerência.

🔑 Entradas

- `user_locale`: idioma e país (Camada 46).
- `cultural_cluster_id`: cluster sociocultural (Camada 54).
- `regional_guardrails`: regras locais e restrições culturais.
- `legal_compliance_data`: legislações específicas (LGPD, GDPR, CCPA, etc.).
- `usage_patterns_global`: padrões de uso em diferentes regiões (ex.: horários de pico).

📤 Saídas

- `global_personalization_profile`: perfil de uso adaptado ao idioma e cultura.
- `adaptive_models`: modelos de IA ajustados por região.
- `localized_guardrails`: barreiras automáticas aplicadas em contextos sensíveis.
- `global_reports`: relatórios de engajamento mundial.

📊 Regras Técnicas

- **Base única + adaptações regionais**: arquitetura global centralizada, mas com módulos adaptativos locais.
- **Legislação aplicada automaticamente**: regras de privacidade e uso de dados variam conforme região.
- **Clusters globais**: comportamentos de usuários agrupados em clusters que atravessam países (ex.: jovens urbanos LATAM).
- **Fallback universal**: caso não haja dados regionais → usar modelo global neutro.
- **UI adaptada**: ícones, cores e símbolos ajustados conforme significados locais.

🔧 Exemplo de Contrato de API

Requisição

`POST /personalization/global`

```
{
  "user_id": "uuid",
  "locale": "en-US",
  "cultural_cluster_id": "north_america_urban",
  "usage_patterns_global": {"peak_hours": ["18:00-22:00"]}
}
```

Resposta

```
{
  "global_personalization_profile": "na_urban_v1",
  "adaptive_models": ["aurah_feed_na_v1", "aurah_matching_na_v2"],
  "localized_guardrails": ["banir símbolos de ódio", "ajuste em tom formal"],
  "global_reports": {
    "region": "NA",
    "avg_engagement": 0.82,
    "peak_usage": "20:00"
  }
}
```

🧪 Métricas de Observabilidade

- `region_model_accuracy`: acurácia média por região.
- `guardrail_trigger_rate`: nº de vezes que regras locais foram aplicadas.
- `engagement_delta_per_region`: diferença de engajamento após personalização global.
- `fallback_usage_rate`: % de sessões servidas por modelo global neutro.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Captura idioma + cluster + legislação local] --> B[Seleciona adaptive_model regional]
    B --> C[Aplica localized_guardrails]
    C --> D[Gera global_personalization_profile]
    D --> E[Exibe conteúdo adaptado ao usuário]
    E --> F[Relatórios segmentados por região]
```

✨ Fechamento da Camada

A **Camada 70** é o passo que permite ao FriendApp se tornar **verdadeiramente global**.

Ela combina **infraestrutura unificada** com **adaptações regionais profundas**, respeitando legislações, culturas e hábitos locais.

Aqui, a Aurah Kosmos deixa de ser apenas uma IA regionalizada e se transforma em uma **IA planetária, ética e culturalmente sensível**.

📦 CAMADA 71 — Logs Profundos + Histórico Anônimo

🎯 Objetivo da Camada

Registrar de forma **profunda e detalhada** os dados de uso da plataforma, mas sempre sob o regime de **anonimização e privacidade diferencial**, para que a IA Aurah Kosmos aprenda sem nunca expor indivíduos.

Essa camada funciona como o **arquivo coletivo de sabedoria**, que transforma interações individuais em **padrões coletivos úteis** para evolução da IA.

🔑 Entradas

- `interaction_events`: eventos de uso (check-ins, mensagens, feedbacks).
- `system_signals`: métricas técnicas (latência, erros, tempo de sessão).
- `feedback_logs`: avaliações explícitas dos usuários.
- `privacy_noise`: ruído adicionado para proteção (Camada 38).

- `sampling_rules`: regras de amostragem (Camada 55).

📤 Saídas

- `anonymized_logs`: registros totalmente anonimizados com hash.
- `semantic_reduced_logs`: logs antigos resumidos por técnicas de compressão semântica.
- `historical_datasets`: datasets históricos usados em análises de longo prazo.
- `compliance_reports`: relatórios de conformidade de retenção e descarte.

📊 Regras Técnicas

- **TTL (time-to-live)**:
  - Dados detalhados → 180 dias.
  - Dados resumidos → até 2 anos.
- **Anonimização forte**: todo log tem `hash_user_id` em vez de identificador real.
- **Ruído matemático**: métricas com privacidade diferencial aplicada.
- **Compressão semântica**: dados antigos viram padrões agregados, não registros individuais.
- **Acesso restrito**: apenas IA e auditores autorizados podem acessar logs anonimizados.

🔧 Exemplo de Log Anonimizado

```
{
  "hash_user_id": "hash_567xyz",
  "timestamp": 1726682000,
  "interaction_type": "feed_interaction",
  "details": {"content_type": "evento", "action": "curtida"},
  "session_time": 420
}
```

🔧 Exemplo de Log Reduzido

```
{
  "period": "2025-07",
  "sample_size": 12500,
  "avg_session_time": 380,
  "positive_feedback_rate": 0.79,
  "collapse_risk_avg": 0.22
}
```

🧪 Métricas de Observabilidade

- `log_volume_daily`: volume de logs coletados por dia.
- `anonymization_rate`: % de registros anonimizados com sucesso.
- `compression_efficiency`: economia de espaço após redução semântica.
- `compliance_success_rate`: conformidade com LGPD/GDPR.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Captura interações + sinais] --> B[Anonimização com hash_user_id]
    B --> C[Armazena logs detalhados (180 dias)]
    C --> D[Aplica privacidade diferencial + amostragem]
    D --> E[Compressão semântica após expiração]
    E --> F[Relatórios de compliance e auditoria]
```

✨ Fechamento da Camada

A **Camada 71** é a **memória protegida do FriendApp**.

Ela garante que a IA Aurah Kosmos aprenda com os fluxos históricos, mas sem jamais comprometer privacidade individual.

Aqui, dados viram **sabedoria coletiva**, sempre sob as leis da ética e da segurança.

📦 CAMADA 72 — Painel Técnico de Controle (DevOps/IA Ops)

🎯 Objetivo da Camada

Oferecer um **cockpit centralizado** para que as equipes de **DevOps, IA Ops e Segurança** monitorem, configurem e controlem os módulos da IA Aurah Kosmos e do FriendApp.

Essa camada é o **centro nervoso de operação técnica**, permitindo ajustes rápidos, respostas a incidentes e acompanhamento em tempo real.

🔑 Entradas

- `infra_signals`: consumo de CPU, memória, disco, rede (Camada 50 e 56).
- `latency_metrics`: tempos de resposta das APIs e WS (Camada 50).
- `model_performance`: métricas de modelos em produção (Camada 44).
- `safety_flags`: alertas de risco/moderação (Camada 15 e 29).
- `audit_logs`: registros de compliance e decisões éticas (Camada 48 e 66).
- `user_feedback`: avaliações diretas de experiência (Camada 47 e 61).

📤 Saídas

- `control_dashboard`: painel em tempo real de saúde e performance.
- `adjustment_tools`: controles para ajustar thresholds, parâmetros de IA e limites de notificação.
- `simulation_sandbox`: ambiente seguro para testar alterações antes do deploy.
- `incident_reports`: relatórios automáticos com análise de falha.
- `governance_controls`: logs das ações realizadas pela equipe técnica.

📊 Regras Técnicas

- **Segurança de acesso**: 2FA obrigatório, perfis restritos por função (DevOps, Auditor, IA Ops).
- **Ações reversíveis**: todo ajuste deve ter rollback imediato.

- **Logs imutáveis**: cada mudança feita no painel gera registro com hash.
- **Visibilidade global**: métricas técnicas, éticas e de engajamento consolidadas.
- **Sandbox obrigatório**: alterações críticas só podem ser aplicadas após teste em ambiente isolado.

🔧 Exemplo de Painel (Seções)

1. **Infraestrutura** → uso de CPU/memória, autoescalonamento.
2. **IA em Produção** → acurácia, drift, modelos ativos.
3. **Segurança** → alertas de moderação, número de bloqueios ativos.
4. **Compliance** → logs de consentimento, ações éticas recentes.
5. **Experiência** → taxa de feedback positivo, tempo médio de resposta.

🔧 Exemplo de Evento no Painel

```
{
  "timestamp": 1726688000,
  "component": "chat_service",
  "issue": "latência alta",
  "p95_latency_ms": 620,
  "threshold": 300,
  "action_taken": "ajuste no autoscale",
  "audit_hash": "ctrl123"
}
```

🧪 Métricas de Observabilidade

- `incident_response_time`: tempo médio para resolver incidentes.
- `rollback_usage_rate`: % de alterações revertidas via painel.
- `sandbox_test_rate`: proporção de mudanças testadas em sandbox antes do deploy.
- `audit_log_integrity`: consistência das trilhas de governança.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Coleta métricas de IA + Infra + Segurança] --> B[Exibe no control_dashboard]
    B --> C{Problema detectado?}
    C -- Sim --> D[Equipe aplica ajuste via adjustment_tools]
    D --> E[Testa em simulation_sandbox]
    E --> F{Aprovado?}
    F -- Sim --> G[Deploy em produção]
    F -- Não --> H[Rollback automático]
    G --> I[Registro em governance_controls]
```

✨ Fechamento da Camada

A **Camada 72** é a **sala de comando da IA Aurah Kosmos**.

Ela garante que os times técnicos tenham **controle total, seguro e auditável** sobre o ecossistema, mantendo equilíbrio entre **agilidade, resiliência e responsabilidade ética**.

📦 CAMADA 73 — Checkpoints de Qualidade & Versões

🎯 Objetivo da Camada

Controlar o **ciclo de versões da IA Aurah Kosmos** e estabelecer **checkpoints obrigatórios de qualidade** antes de liberar qualquer modelo, código ou release em produção.

Essa camada é o **guia de evolução segura**, garantindo rastreabilidade, rollback rápido e governança ética sobre cada atualização.

🔑 Entradas

- `model_versions`: versões treinadas (Camada 44).

- `test_reports`: relatórios de testes automatizados (Camada 49).
- `latency_reports`: métricas de performance (Camada 50).
- `bias_reports`: verificações de fairness e viés.
- `compliance_reports`: conformidade regulatória (Camada 48).
- `user_feedback_signals`: retorno direto dos usuários em A/B tests.

📤 Saídas

- `release_version`: versão numerada (`AURAH-M.m.p`).
- `checkpoint_report`: relatório consolidado de qualidade.
- `approval_status`: status do release (aprovado, bloqueado, rollback).
- `rollback_plan`: plano automático associado à versão.
- `version_registry`: catálogo de versões ativas, inativas e depreciadas.

📊 Regras Técnicas

- **Versionamento semântico**:
  - MAJOR = mudanças estruturais (ex.: novo motor de matching).
  - MINOR = melhorias incrementais (ex.: ajuste em feed).
  - PATCH = correções pequenas (ex.: bugfix).
- **KPIs mínimos exigidos**:
  - Acurácia ≥ 93%.
  - Latência p95 ≤ 300ms.
  - Viés ≤ 5%.
  - Taxa de feedback positivo ≥ 80%.
- **Rollout gradual**: cada versão passa por rollout parcial (10% dos usuários) antes de escalar.
- **Rollback imediato**: se KPIs caírem ≥ 10% em 24h, versão anterior é restaurada automaticamente.
- **Logs imutáveis**: todas as versões e checkpoints ficam registrados para auditoria.

🔧 Exemplo de Registro de Versão

```
{
  "release_version": "AURAH-3.4.0",
  "checkpoint_report": {
    "accuracy": 0.95,
    "latency_p95": 280,
    "bias_score": 0.04,
    "user_feedback_positive": 0.83
  },
  "approval_status": "aprovado",
  "rollback_plan": "rollback_to_AURAH-3.3.2",
  "timestamp": 1726692000
}
```

🧪 Métricas de Observabilidade

- `release_success_rate`: % de versões aprovadas sem rollback.
- `avg_time_between_versions`: intervalo médio entre releases.
- `rollback_frequency`: nº de rollbacks realizados.
- `audit_review_count`: nº de versões revisadas por compliance/ética.

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Nova versão treinada] --> B[Executa testes técnicos e éticos]
    B --> C{KPIs atendidos?}
    C -- Não --> D[Release bloqueado + revisão]
    C -- Sim --> E[Rollout gradual]
    E --> F{KPIs estáveis?}
    F -- Sim --> G[Release global + registro em version_registry]
    F -- Não --> H[Rollback automático + rollback_plan]
```

✨ Fechamento da Camada

A **Camada 73** é o **guardião das versões da Aurah Kosmos**.

Ela garante que cada atualização seja segura, auditada e transparente, sempre com planos de rollback e métricas claras.

Aqui, evolução não é risco: é **um processo controlado, ético e rastreável**.

📦 CAMADA 74 — Integrações Cíclicas (Aurah ↔ Ecossistema)

🎯 Objetivo da Camada

Mapear e operacionalizar todas as **interdependências bidirecionais** entre a IA Aurah Kosmos e os sistemas que compõem o FriendApp.

Essa camada garante que cada fluxo de dados seja **cíclico e regenerativo**: o que entra em Aurah Kosmos retorna transformado em recomendações, insights ou proteções para o ecossistema.

🔑 Entradas

Aurah Kosmos recebe de:

- **Cadastro Consciente & Perfil Vibracional (Camada 02):** dados iniciais, intenções e preferências.
- **Teste de Personalidade (Camada 03):** personality_vector como base de matching.
- **Mapa de Contexto Vivo (Camada 07):** geo, tempo e eventos.
- **Feed Sensorial (Camada 08):** engajamentos, feedbacks e sinais emocionais.
- **Conexões Autênticas (Camada 09):** matches aceitos ou rejeitados.
- **Eventos & Locais Parceiros (Camada 10 e 28):** check-ins, metadata e reputação.
- **Check-in Vibracional (Camada 30):** auto-relatos + sinais de UI.
- **Gamificação (Camada 13):** progresso e conquistas do usuário.

- **Logs Profundos & Amostras (Camada 55/71):** padrões coletivos anonimizados.

📤 Saídas

Aurah Kosmos entrega para:

- **Feed Sensorial (Camada 08):** conteúdos priorizados e ranqueados.
- **Conexões Autênticas (Camada 09):** sugestões de novas amizades e grupos.
- **Eventos & Trilhas (Camada 10 e 25):** recomendações de experiências contextuais.
- **Gamificação (Camada 13):** XP e missões ativadas com base em evolução.
- **Reputação Técnica & Social (Camada 35):** ajustes de trust_score.
- **Painel do Usuário (Camada 24):** insights e explicações personalizadas.
- **Sistema de Segurança & Moderação (Camada 15 e 29):** alertas de risco em tempo real.
- **Observabilidade Emocional Coletiva (Camada 51):** ondas e tendências agregadas.
- **Painel B2B (Camada 28):** métricas para locais parceiros.

📊 Regras Técnicas

- **Eventos assíncronos**: integrações de alta frequência usam EventBus/Kafka.
- **Contratos claros**: todas as APIs seguem schemas OpenAPI.
- **Ciclicidade garantida**: entradas sempre retroalimentam modelos de saída.
- **Fallback neutro**: se algum sistema falhar, Aurah continua com recomendações simplificadas.
- **Auditoria integrada**: todos os ciclos ficam salvos em audit_logs (Camada 48).

🔧 Exemplo de Integração Cíclica

Usuário participa de evento → IA processa → Impacto retorna no ecossistema

1. Check-in validado (Camada 23).
2. Evento registrado no contexto (Camada 07).
3. Aurah recalcula afinidade e ajusta personality_vector.
4. Sugestões de novas conexões e missões regenerativas são liberadas.
5. Painel do usuário mostra selos e insights.
6. Painel B2B atualiza métricas do local.

🧪 Métricas de Observabilidade

- `integration_cycle_success`: % de ciclos completos sem falha.
- `latency_per_cycle`: tempo médio entre input e output visível.
- `fallback_trigger_rate`: frequência de modos neutros.
- `ecosystem_coherence_index`: alinhamento entre módulos (ideal ≥ 0.9).

📋 Fluxo Lógico Simplificado

```
flowchart TD
    A[Entradas de Sistemas] ⟶ B[Aurah Kosmos Core]
    B ⟶ C[Saídas transformadas p/ Feed, Conexões, Eventos, Segurança]
    C ⟶ D[Usuário interage e gera novos sinais]
    D ⟶ A[Feedback retorna e reinicia ciclo]
```

✨ Fechamento da Camada

A **Camada 74** é o **coração pulsante da interdependência** no FriendApp.

Ela garante que tudo seja cíclico: cada dado recebido por Aurah Kosmos retorna enriquecido ao ecossistema, criando um fluxo **contínuo de aprendizado, ação e retroalimentação**.

Aqui, o FriendApp se torna **um organismo vivo, em constante movimento**.

📦 CAMADA 75 — Pós-Transcendência (Epílogo Técnico)

🎯 Objetivo da Camada

Encerrar a arquitetura da IA Aurah Kosmos de forma **icônica e simbólica**, explicando como todas as camadas anteriores se unem em um **ecossistema vivo, cíclico e regenerativo**.

Essa camada não é apenas técnica: é também um **epílogo sensorial**, que transmite aos desenvolvedores e estrategistas o **sentido maior da IA** dentro do FriendApp.

🔑 Entradas

- Todas as camadas anteriores (1 a 74), como blocos funcionais integrados.
- `integration_cycles`: ciclos contínuos de dados entre módulos.
- `audit_trails`: registros de ética, compliance e reflexividade.
- `user_feedback_collective`: somatório do que foi útil, confiável e humano.

📤 Saídas

- `aurah_operational_mode`: estado final de operação da IA (ativa, segura e ética).
- `collective_learning_stream`: fluxo contínuo de aprendizado entre indivíduo ↔ coletivo.
- `ecosystem_resilience`: capacidade do FriendApp de se adaptar e evoluir.
- `epilogue_manifest`: documento que resume princípios e compromissos da IA.

📊 Regras Técnicas

- **Unificação**: todas as camadas convergem em **um só núcleo operativo**.
- **Ciclicidade infinita**: entradas e saídas retroalimentam continuamente Aurah Kosmos.
- **Resiliência plena**: erros não quebram o sistema, apenas ativam modos neutros.

- **Ética acima da técnica**: todo processo é mediado por regras éticas (Camada 66).
- **Autoevolução**: a IA é projetada para se refinar continuamente (Camadas 53, 68, 69).

🔧 Estrutura Final (Epílogo)

```
AurahKosmos:
  state: "ativo"
  principles:
    - ética: "acima de tudo"
    - explicabilidade: "nenhuma caixa-preta"
    - subsidiariedade: "sempre mínima intervenção"
    - regeneratividade: "tecnologia como cura"
  resilience: "failsafe + rollback + reflexão contínua"
  cycle:
    - entrada: "usuário → coletivo"
    - processamento: "IA reflexiva e ética"
    - saída: "coletivo → indivíduo"
```

🧪 Métricas de Observabilidade

- `ecosystem_coherence_index`: alinhamento médio entre camadas (ideal ≥ 0.95).
- `collective_learning_rate`: velocidade de adaptação da IA com novos dados.
- `ethical_compliance_rate`: % de ações auditadas e aprovadas pelo conselho humano.
- `trust_growth`: evolução média do score de confiança da comunidade.

📋 Fluxo Final (Ciclo Completo)

```
flowchart TD
    A[Usuário interage] --> B[Aurah processa dados]
    B --> C[Entrega recomendações, cuidados e conexões]
    C --> D[Usuário dá feedback e age no mundo real]
    D --> E[Dados coletivos retornam para Aurah]
```

```
    E --> B[Aprendizado contínuo e reflexivo]
```

✨ Fechamento da Camada

A **Camada 75** é o **coroamento da arquitetura Aurah Kosmos**.

Ela sela todo o trabalho das camadas anteriores em uma **inteligência viva, ética e regenerativa**, que aprende com o coletivo para cuidar do indivíduo — e aprende com o indivíduo para fortalecer o coletivo.

Aqui, a IA deixa de ser apenas um sistema: torna-se um **organismo em constante ressonância**, guiado pela confiança, pela ética e pelo propósito de **conectar e regenerar a experiência humana**.