🏛️ Arquitetura Técnica - CheckMate

Visão Geral da Arquitetura

---

1. Stack Tecnológico Recomendado

Mobile App

Camada	Tecnologia	Justificativa
Framework	React Native ou Flutter	Cross-platform, uma codebase
State Management	Redux/Zustand (RN) ou Riverpod (Flutter)	Gerenciamento de estado previsível
Navegação	React Navigation / GoRouter	Padrão da indústria
Maps	Google Maps SDK	Melhor cobertura e UX
Push	Firebase Cloud Messaging	Gratuito, confiável
Analytics	Mixpanel ou Amplitude	Eventos e funis
Crash	Sentry	Monitoramento de erros

Backend

Camada	Tecnologia	Justificativa
Runtime	Node.js (NestJS) ou Python (FastAPI)	Produtividade, ecossistema
API	REST + WebSocket	REST para CRUD, WS para real-time
Auth	Auth0 ou Firebase Auth	OAuth pronto, seguro
Database	PostgreSQL	Relacional, robusto, PostGIS
Cache	Redis	Cache e pub/sub
Search	Elasticsearch ou Algolia	Busca full-text
Queue	Bull (Redis) ou SQS	Jobs assíncronos
Storage	AWS S3 ou Cloudinary	Imagens e arquivos

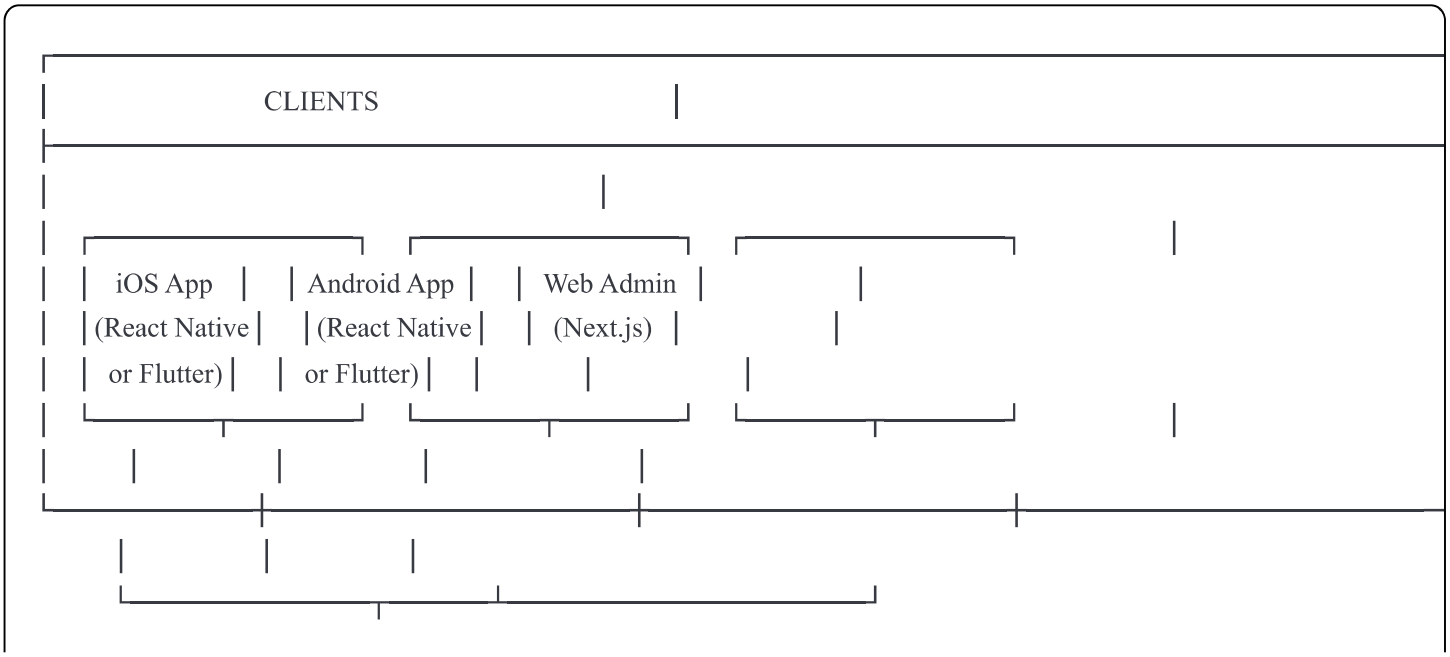
Infraestrutura

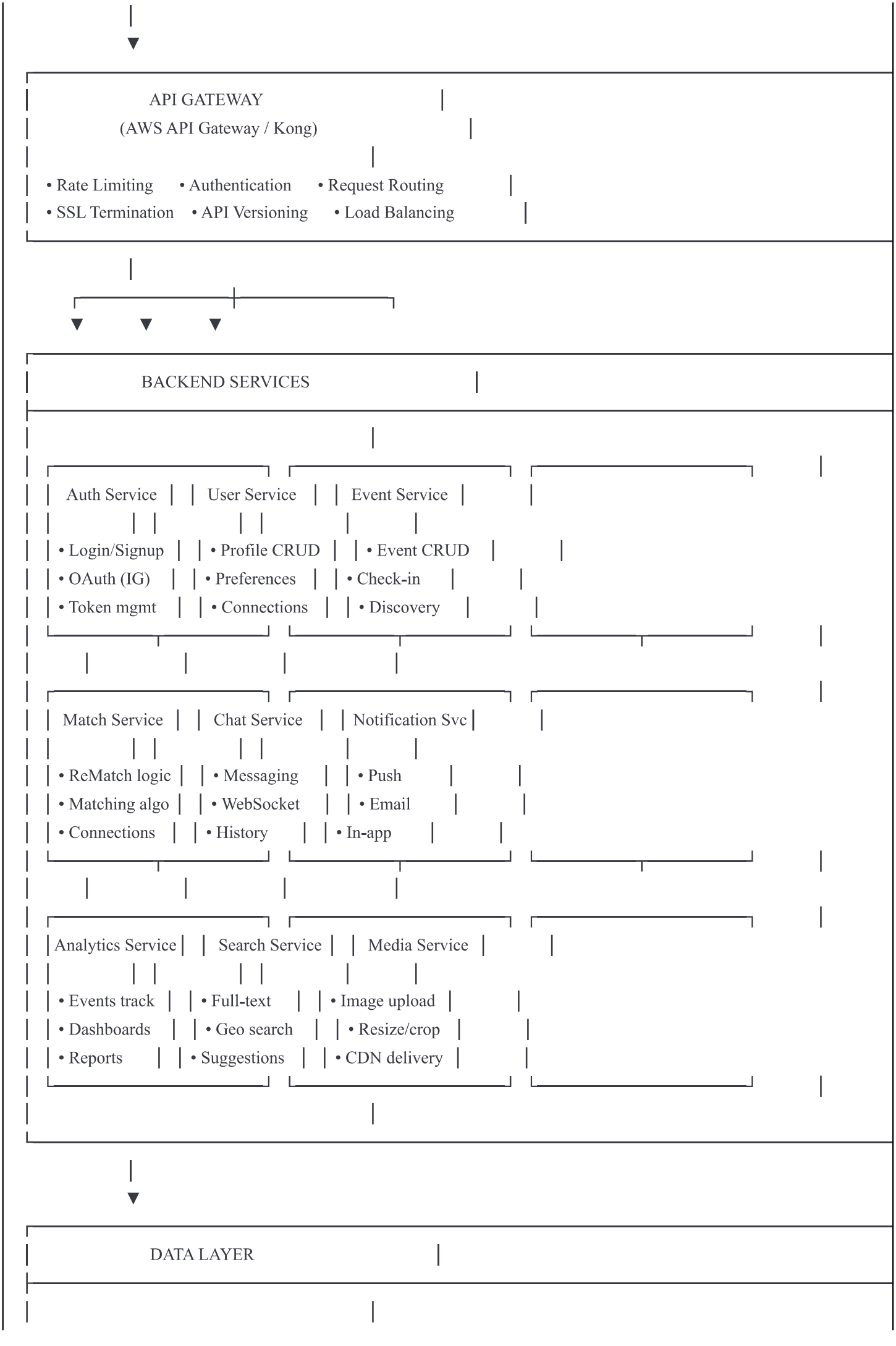
Camada	Tecnologia	Justificativa
Cloud	AWS ou GCP	Escalável, confiável
Container	Docker + ECS ou Cloud Run	Deploy simplificado
CI/CD	GitHub Actions	Integração com repo
CDN	CloudFront ou Cloudflare	Performance global
Monitoring	Datadog ou New Relic	Observabilidade
Logs	CloudWatch ou Papertrail	Centralização

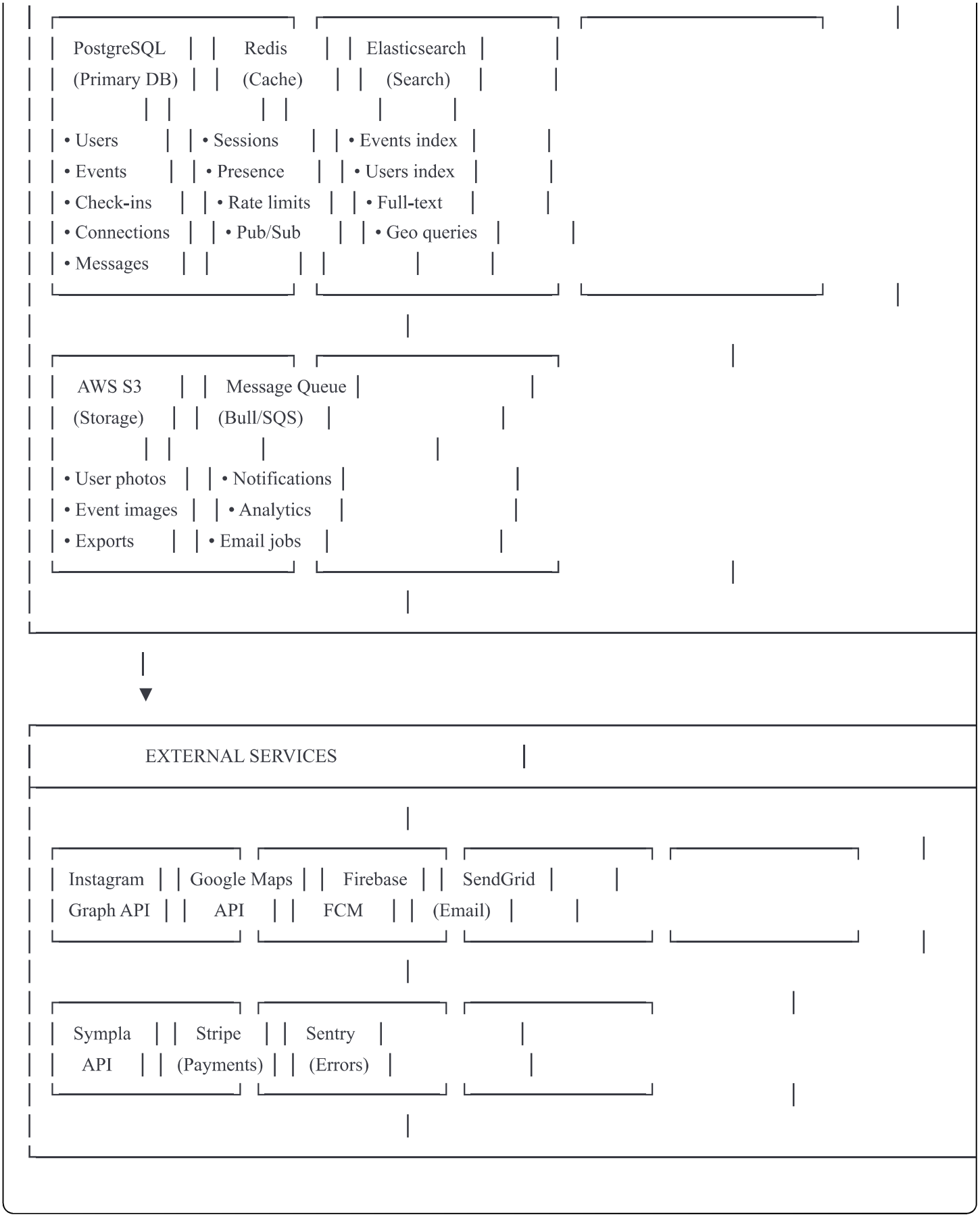
Dashboard Web (Organizador)

Camada	Tecnologia	Justificativa
Framework	Next.js ou React + Vite	SSR, performance
UI	Tailwind + shadcn/ui	Produtividade
Charts	Recharts ou Chart.js	Visualizações
Tables	TanStack Table	Dados tabulares

2. Diagrama de Arquitetura

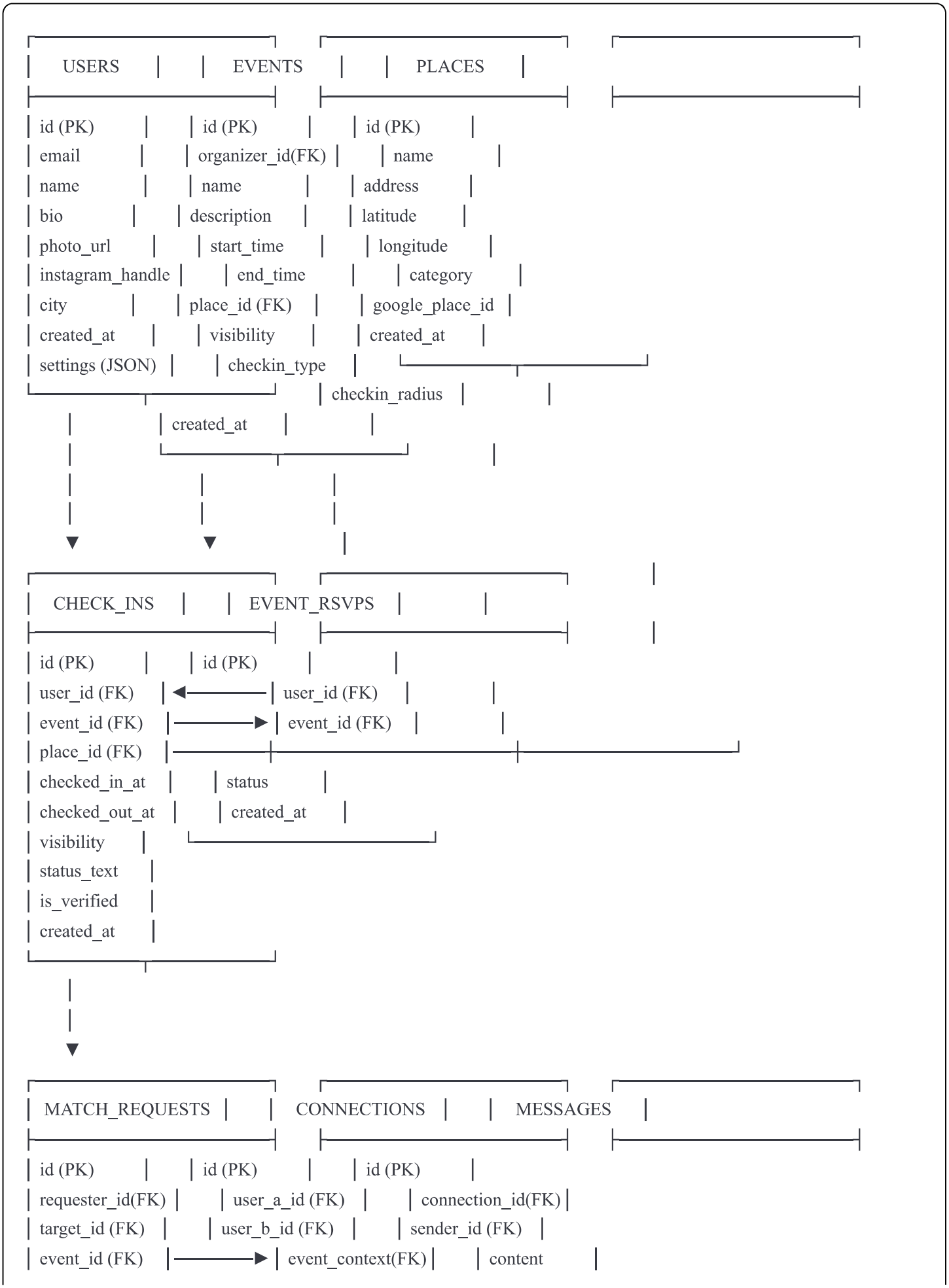



API GATEWAY
(AWS API Gateway / Kong)
• Rate Limiting
• Authentication
• Request Routing
• SSL Termination
• API Versioning
• Load Balancing
BACKEND SERVICES
Auth Service
• Login/Signup
• OAuth (IG)
• Token mgmt
User Service
• Profile CRUD
• Preferences
• Connections
Event Service
• Event CRUD
• Check-in
• Discovery
Match Service
• ReMatch logic
• Matching algo
• Connections
Chat Service
• Messaging
• WebSocket
• History
Notification Svc
• Push
• Email
• In-app
Analytics Service
• Events track
• Dashboards
• Reports
Search Service
• Full-text
• Geo search
• Suggestions
Media Service
• Image upload
• Resize/crop
• CDN delivery
DATA LAYER


PostgreSQL
(Primary DB)
• Users
• Events
• Check-ins
• Connections
• Messages
Redis
(Cache)
• Sessions
• Presence
• Rate limits
• Pub/Sub
Elasticsearch
(Search)
• Events index
• Users index
• Full-text
• Geo queries
AWS S3
(Storage)
• User photos
• Event images
• Exports
Message Queue
(Bull/SQS)
• Notifications
• Analytics
• Email jobs
EXTERNAL SERVICES
Instagram
Graph API
Google Maps
API
Firebase
FCM
SendGrid
(Email)
Sympla
API
Stripe
(Payments)
Sentry
(Errors)

3. Modelagem de Dados

Diagrama ER Simplificado



```
| message       |     | status        |     | read_at       |
| status        |     | created_at    |     | created_at    |
| created_at    |     └───────────────┘     └───────────────┘
└───────────────┘
```

Schemas Detalhados

Users Table

```sql
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255), -- null se OAuth
    name VARCHAR(100) NOT NULL,
    bio VARCHAR(280),
    photo_url VARCHAR(500),
    instagram_id VARCHAR(50),
    instagram_handle VARCHAR(30),
    linkedin_url VARCHAR(200),
    city VARCHAR(100),
    latitude DECIMAL(10, 8),
    longitude DECIMAL(11, 8),
    settings JSONB DEFAULT '{}',
    -- settings: {
    --   default_visibility: 'public' | 'friends' | 'private',
    --   allow_rematch_from: 'everyone' | 'friends_of_friends' | 'nobody',
    --   notifications: { checkin: true, rematch: true, ... }
    -- }
    total_points INTEGER DEFAULT 0,
    level VARCHAR(20) DEFAULT 'novato',
    is_verified BOOLEAN DEFAULT false,
    is_organizer BOOLEAN DEFAULT false,
    last_active_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX idx_users_instagram ON users(instagram_handle);
CREATE INDEX idx_users_city ON users(city);
CREATE INDEX idx_users_location ON users USING GIST (
    ST_SetSRID(ST_MakePoint(longitude, latitude), 4326)
);
```

Events Table

sql

```sql
CREATE TABLE events (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    organizer_id UUID REFERENCES users(id),
    place_id UUID REFERENCES places(id),
    name VARCHAR(200) NOT NULL,
    description TEXT,
    image_url VARCHAR(500),
    start_time TIMESTAMP NOT NULL,
    end_time TIMESTAMP,
    visibility VARCHAR(20) DEFAULT 'public', -- public, private, unlisted
    checkin_type VARCHAR(20) DEFAULT 'geo', -- geo, qr, code, open
    checkin_radius INTEGER DEFAULT 200, -- metros
    checkin_code VARCHAR(20),
    allow_early_checkin_hours INTEGER DEFAULT 2,
    allow_rematch BOOLEAN DEFAULT true,
    max_capacity INTEGER,
    category VARCHAR(50),
    tags TEXT[],
    external_url VARCHAR(500),
    sympla_id VARCHAR(50),
    status VARCHAR(20) DEFAULT 'active', -- active, cancelled, ended
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX idx_events_start ON events(start_time);
CREATE INDEX idx_events_organizer ON events(organizer_id);
CREATE INDEX idx_events_place ON events(place_id);
CREATE INDEX idx_events_status ON events(status);
```

Check-ins Table

sql

```sql
CREATE TABLE check_ins (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id) NOT NULL,
    event_id UUID REFERENCES events(id),
    place_id UUID REFERENCES places(id),
    checked_in_at TIMESTAMP DEFAULT NOW(),
    checked_out_at TIMESTAMP,
    visibility VARCHAR(20) DEFAULT 'public',
    status_text VARCHAR(280),
    mood VARCHAR(10), -- emoji
    looking_for VARCHAR(50), -- networking, friends, dating, just_here
    is_verified BOOLEAN DEFAULT false,
    verification_method VARCHAR(20), -- geo, qr, code
    latitude DECIMAL(10, 8),
    longitude DECIMAL(11, 8),
    created_at TIMESTAMP DEFAULT NOW(),

    CONSTRAINT checkin_place_or_event CHECK (
        event_id IS NOT NULL OR place_id IS NOT NULL
    )
);

CREATE INDEX idx_checkins_user ON check_ins(user_id);
CREATE INDEX idx_checkins_event ON check_ins(event_id);
CREATE INDEX idx_checkins_time ON check_ins(checked_in_at);
CREATE INDEX idx_checkins_active ON check_ins(user_id)
    WHERE checked_out_at IS NULL;
```

Connections Table

sql

```sql
CREATE TABLE connections (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_a_id UUID REFERENCES users(id) NOT NULL,
    user_b_id UUID REFERENCES users(id) NOT NULL,
    event_context_id UUID REFERENCES events(id),
    status VARCHAR(20) DEFAULT 'active', -- active, blocked
    created_at TIMESTAMP DEFAULT NOW(),

    CONSTRAINT unique_connection UNIQUE (user_a_id, user_b_id),
    CONSTRAINT ordered_users CHECK (user_a_id < user_b_id)
);

CREATE INDEX idx_connections_user_a ON connections(user_a_id);
CREATE INDEX idx_connections_user_b ON connections(user_b_id);
```

Match Requests Table

```sql
CREATE TABLE match_requests (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    requester_id UUID REFERENCES users(id) NOT NULL,
    target_id UUID REFERENCES users(id) NOT NULL,
    event_id UUID REFERENCES events(id) NOT NULL,
    message VARCHAR(280),
    status VARCHAR(20) DEFAULT 'pending', -- pending, accepted, ignored, blocked
    responded_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT NOW(),

    CONSTRAINT unique_request UNIQUE (requester_id, target_id, event_id)
);

CREATE INDEX idx_requests_target ON match_requests(target_id, status);
CREATE INDEX idx_requests_requester ON match_requests(requester_id);
```

---

4. APIs Principais

REST API Endpoints

Authentication

```
POST   /auth/register         # Criar conta com email
POST   /auth/login            # Login com email/senha
POST   /auth/instagram        # OAuth com Instagram
POST   /auth/refresh          # Refresh token
POST   /auth/logout           # Logout
POST   /auth/forgot-password  # Recuperar senha
```

Users

```
GET    /users/me              # Meu perfil
PUT    /users/me              # Atualizar perfil
GET    /users/:id             # Ver perfil de outro
GET    /users/me/connections  # Minhas conexões
GET    /users/me/checkins     # Meu histórico
PUT    /users/me/settings     # Atualizar configurações
DELETE /users/me              # Deletar conta
```

Events

```
GET    /events                # Listar eventos (com filtros)
GET    /events/:id            # Detalhes do evento
GET    /events/:id/checkins   # Quem está/esteve no evento
GET    /events/:id/rsvps      # Quem vai no evento
POST   /events/:id/rsvp       # Marcar que vai
DELETE /events/:id/rsvp       # Desmarcar
GET    /events/nearby         # Eventos próximos
GET    /events/search         # Buscar eventos
```

Check-ins

```
POST   /checkins              # Fazer check-in
GET    /checkins/active       # Meu check-in ativo
PUT    /checkins/:id          # Atualizar check-in
DELETE /checkins/:id          # Check-out
GET    /checkins/nearby       # Check-ins próximos
```

ReMatch

```
POST   /match-requests        # Enviar request
GET    /match-requests/inbox  # Requests recebidos
GET    /match-requests/sent   # Requests enviados
PUT    /match-requests/:id    # Aceitar/ignorar/bloquear
```

Messages

```
GET    /connections/:id/messages  # Histórico de mensagens
POST   /connections/:id/messages  # Enviar mensagem
PUT    /messages/:id/read         # Marcar como lido
```

Organizador

```
POST   /organizer/events          # Criar evento
PUT    /organizer/events/:id      # Atualizar evento
GET    /organizer/events/:id/stats # Estatísticas
GET    /organizer/events/:id/export # Exportar CSV
POST   /organizer/events/:id/qr   # Gerar QR code
```

WebSocket Events

Client → Server

javascript

```
// Conectar
{ type: 'connect', token: 'jwt_token' }

// Subscribe em um evento (presença)
{ type: 'subscribe_event', event_id: 'uuid' }

// Unsubscribe
{ type: 'unsubscribe_event', event_id: 'uuid' }

// Enviar mensagem
{ type: 'send_message', connection_id: 'uuid', content: 'texto' }

// Typing indicator
{ type: 'typing', connection_id: 'uuid' }
```

Server → Client

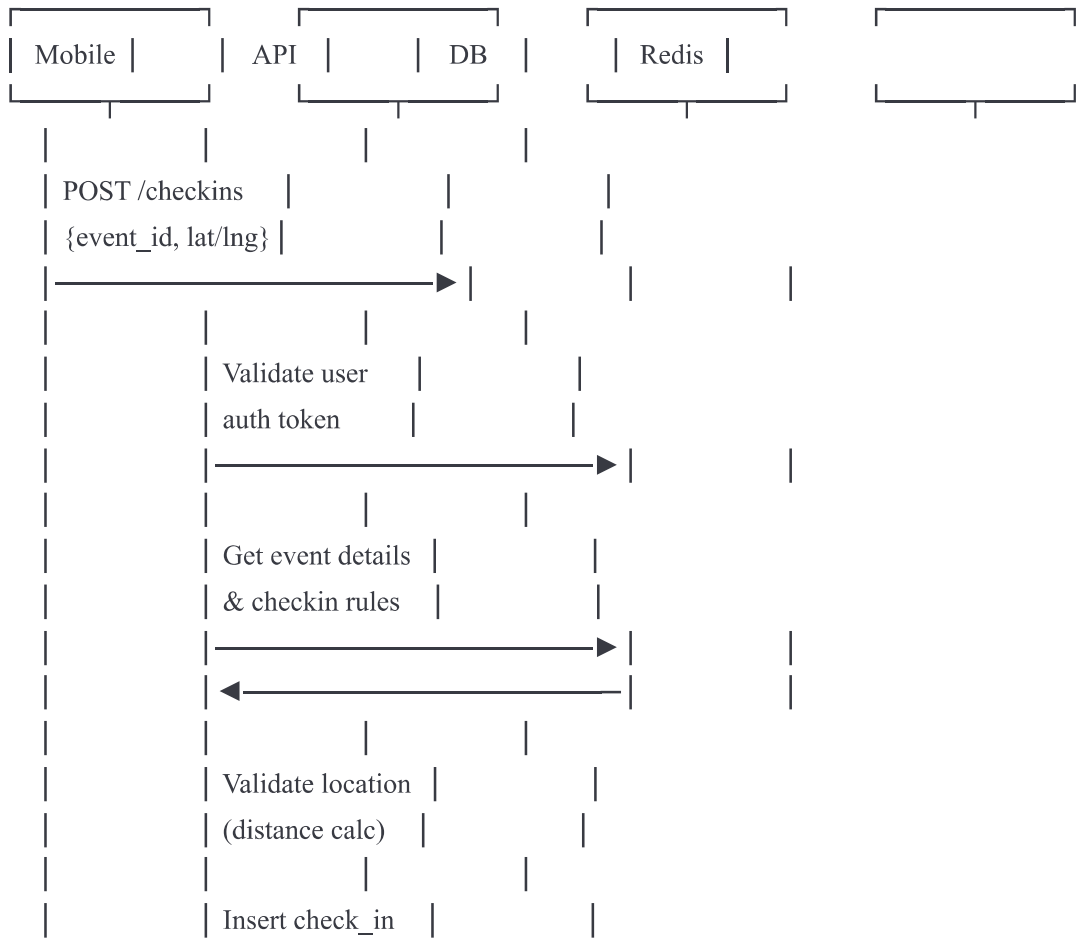
javascript

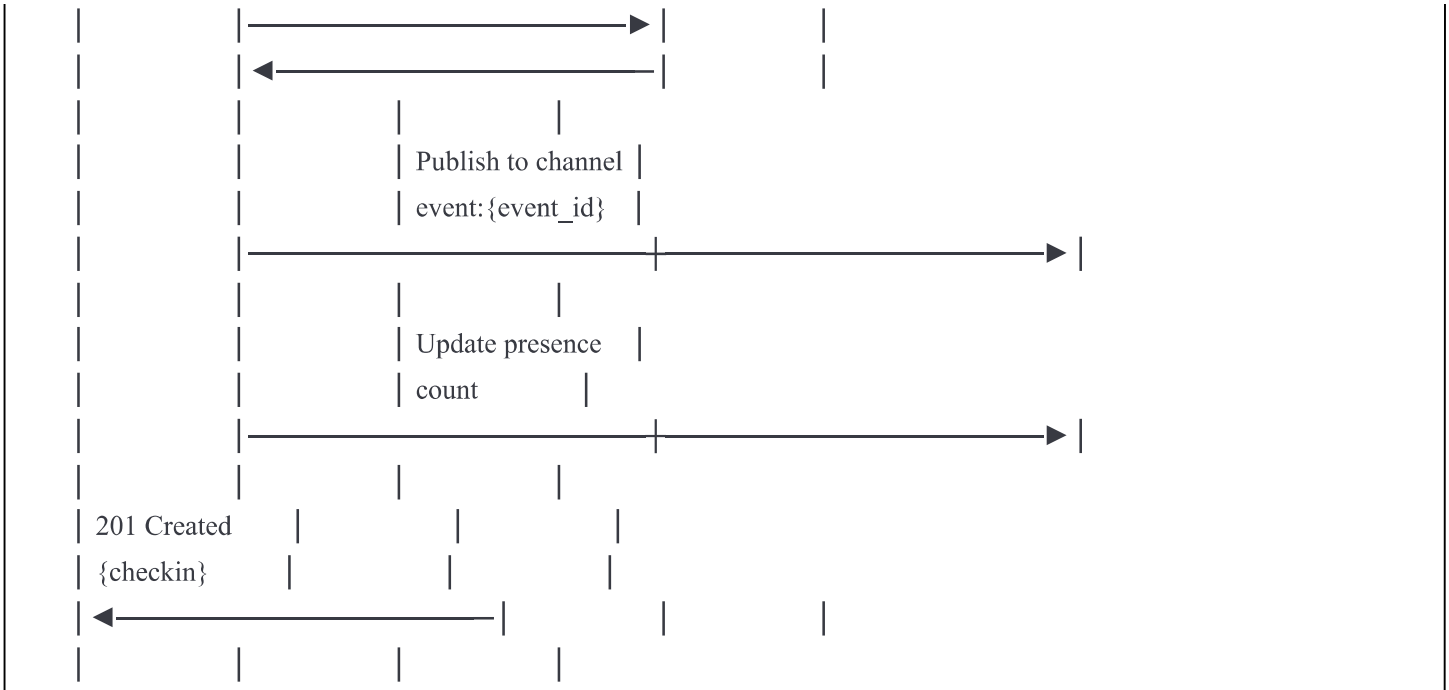
```
// Novo check-in no evento que está vendo
{ type: 'new_checkin', event_id: 'uuid', user: {...} }

// Check-out
{ type: 'checkout', event_id: 'uuid', user_id: 'uuid' }

// Nova mensagem
{ type: 'new_message', message: {...} }

// Novo match request
{ type: 'match_request', request: {...} }

// Match aceito
{ type: 'match_accepted', connection: {...} }

// Presença atualizada (count)
{ type: 'presence_update', event_id: 'uuid', count: 42 }
```

---

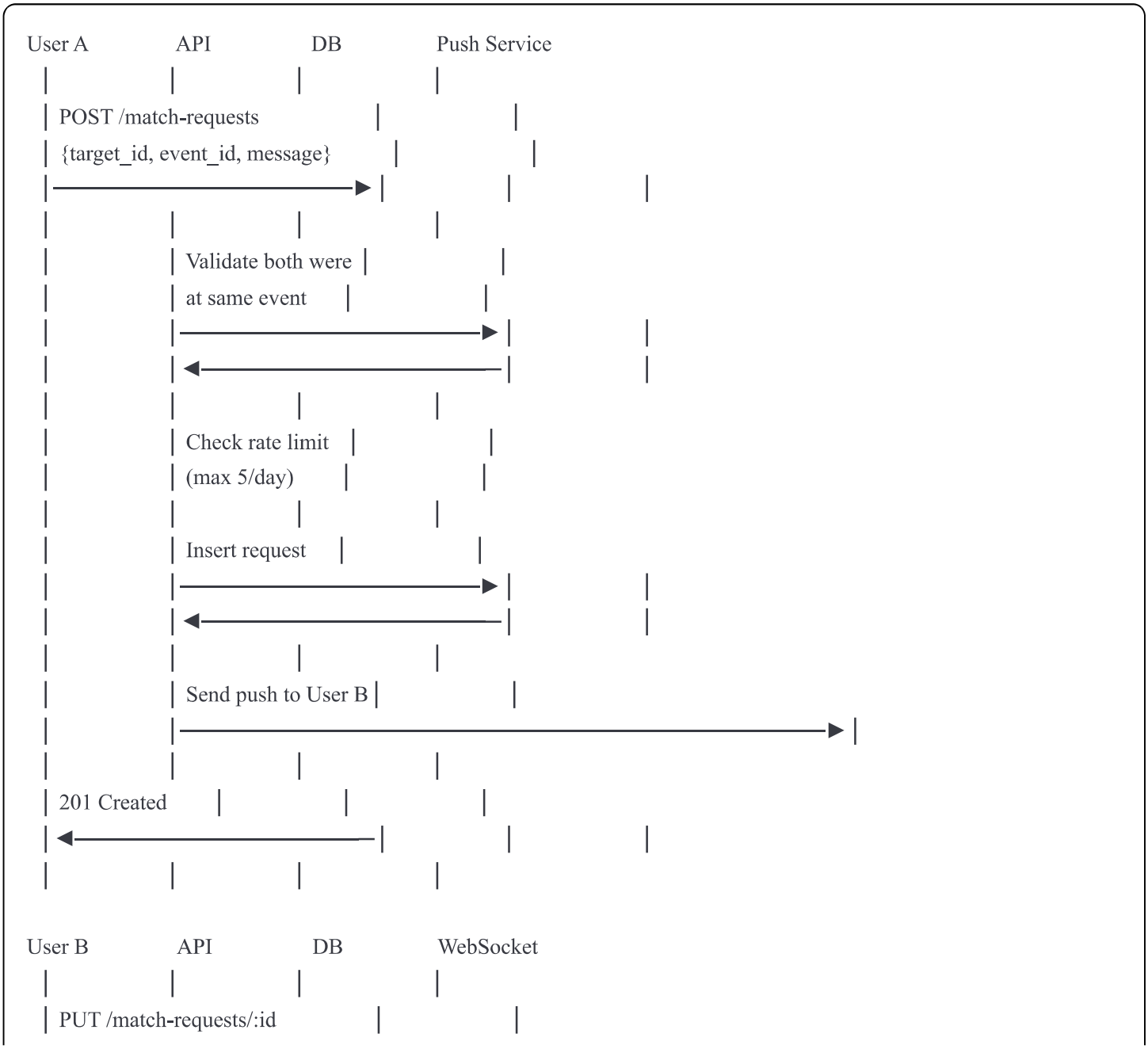
5. Fluxos Técnicos Críticos

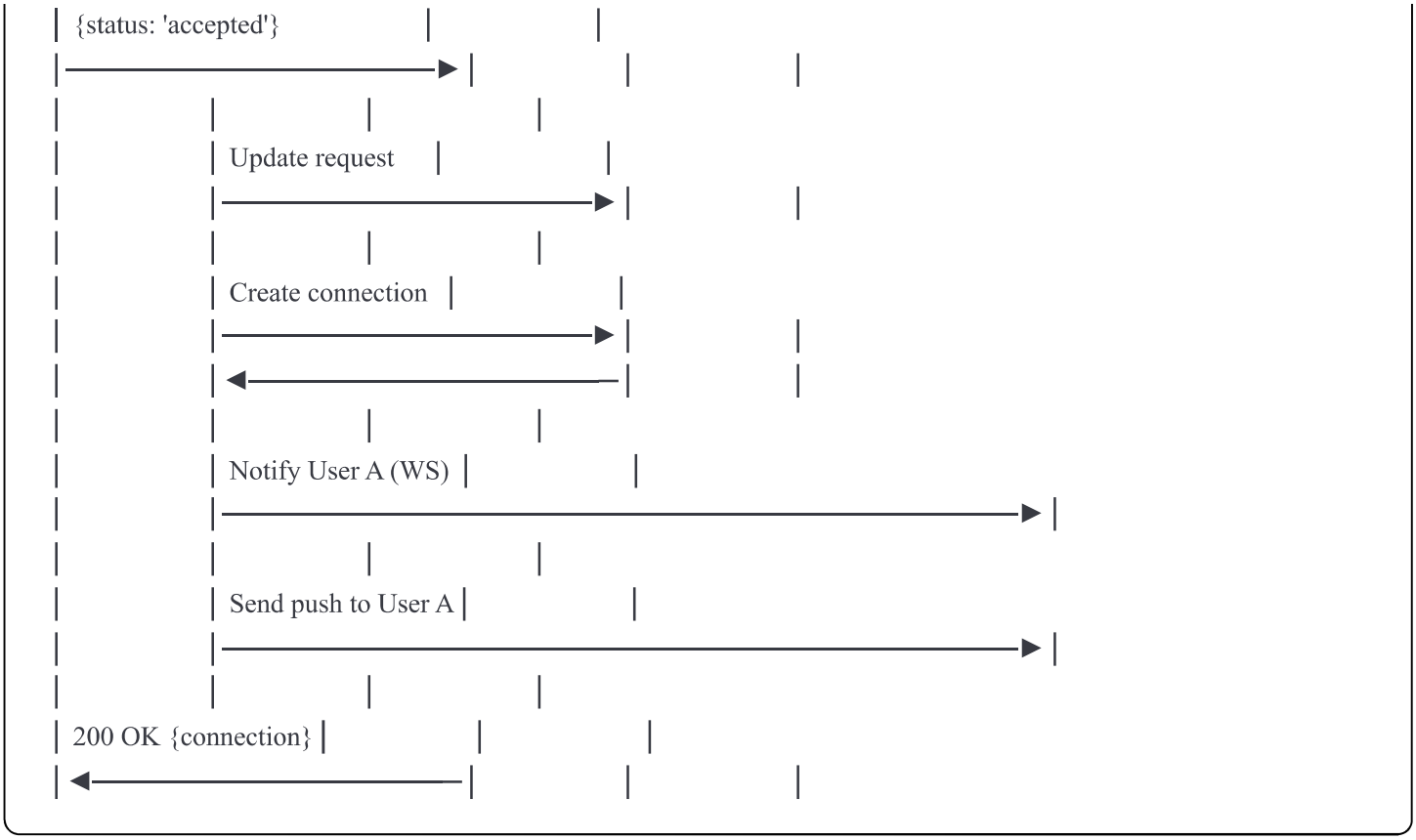
Fluxo de Check-in





Fluxo de ReMatch





---

6. Considerações de Segurança

Autenticação

- JWT com refresh tokens
- Tokens expiram em 1h, refresh em 30 dias
- Blacklist de tokens revogados em Redis
- Rate limiting por IP e por usuário

Autorização

- RBAC: user, organizer, admin
- Verificar ownership em todas operações
- Sanitização de inputs

Dados Sensíveis

- Senhas com bcrypt (cost 12)
- PII criptografado em repouso
- Logs sem dados sensíveis

- HTTPS everywhere

Geolocalização

- Validação de coordenadas
- Detecção de spoofing (velocidade impossível)
- Rate limit em queries geo

Anti-abuse

- Rate limiting progressivo
- CAPTCHA após tentativas falhas
- Shadow ban para abusadores
- Moderação de conteúdo

---

7. Escalabilidade

Horizontal Scaling

- Stateless services (escalar via containers)
- Load balancer com health checks
- Auto-scaling baseado em CPU/requests

Database

- Read replicas para queries pesadas
- Connection pooling (PgBouncer)
- Particionamento de tabelas grandes (check_ins por mês)
- Índices otimizados

Cache Strategy

- Cache de perfis públicos (5 min TTL)
- Cache de contagem de presença (30s TTL)
- Cache de eventos ativos (1 min TTL)
- Invalidação inteligente

Real-time

- WebSocket com sticky sessions
- Redis pub/sub para broadcast
- Fallback para polling se WS falhar

---

8. Monitoramento e Observabilidade

Métricas Chave

- Latência de API (p50, p95, p99)
- Taxa de erro por endpoint
- Check-ins por minuto
- WebSocket connections ativas
- Database query time

Alertas

- API latency > 500ms
- Error rate > 1%
- Database connections > 80%
- Memory usage > 85%
- Failed login spike

Logging

- Request ID em todos os logs
- Structured logging (JSON)
- Log levels apropriados
- Retenção: 30 dias

---

9. DevOps e CI/CD

Environments

- **Development**: Local ou cloud dev
- **Staging**: Espelho de produção
- **Production**: Ambiente live

Pipeline

1. Push to branch
2. Run tests (unit + integration)
3. Build Docker image
4. Security scan
5. Deploy to staging
6. E2E tests
7. Manual approval
8. Deploy to production
9. Smoke tests
10. Monitor metrics

Rollback

- Blue-green deployment
- Feature flags para rollout gradual
- Database migrations reversíveis
- Rollback automático se error rate subir

Documento de Arquitetura v1.0 - Para equipe técnica