掼蛋全球游戏平台 - 完整技术架构设计书 V4.0

Table of Contents

- 执行摘要
- 一、完整系统架构
- 二、前端架构 (Skia 统一)
- 三、后端架构详解
- 核心表结构
    - 四、完整的部署与监控
- .github/workflows/deploy.yml
- 使用 CloudWatch 监控关键指标
- 自定义指标
- 关键告警规则
    - 五、完整的API规范
- OpenAPI 3.0 规范
    - 六、安全与合规
    - 七、性能优化与扩展性
- AWS Auto Scaling 配置
- Lambda 自动扩展
- DynamoDB 自动扩展
- ElastiCache (如果需要)
- RDS (用于分析, 可选)
    - 总结

Skia 统一 + AWS Serverless + Web3 集成方案

版本: 4.0（GameFi 完整版）
日期: 2025年12月
目标: 全球化、跨平台、链上数据、完全去中心化架构

执行摘要

本文档是掼蛋游戏平台的**最终生产级技术架构**，覆盖：

- **前端**: React Native Skia 统一渲染 (Web/iOS/Android) + Web3 钱包集成
- **后端**: AWS Lambda 微服务 + DynamoDB 离链数据 + Polygon 链上数据

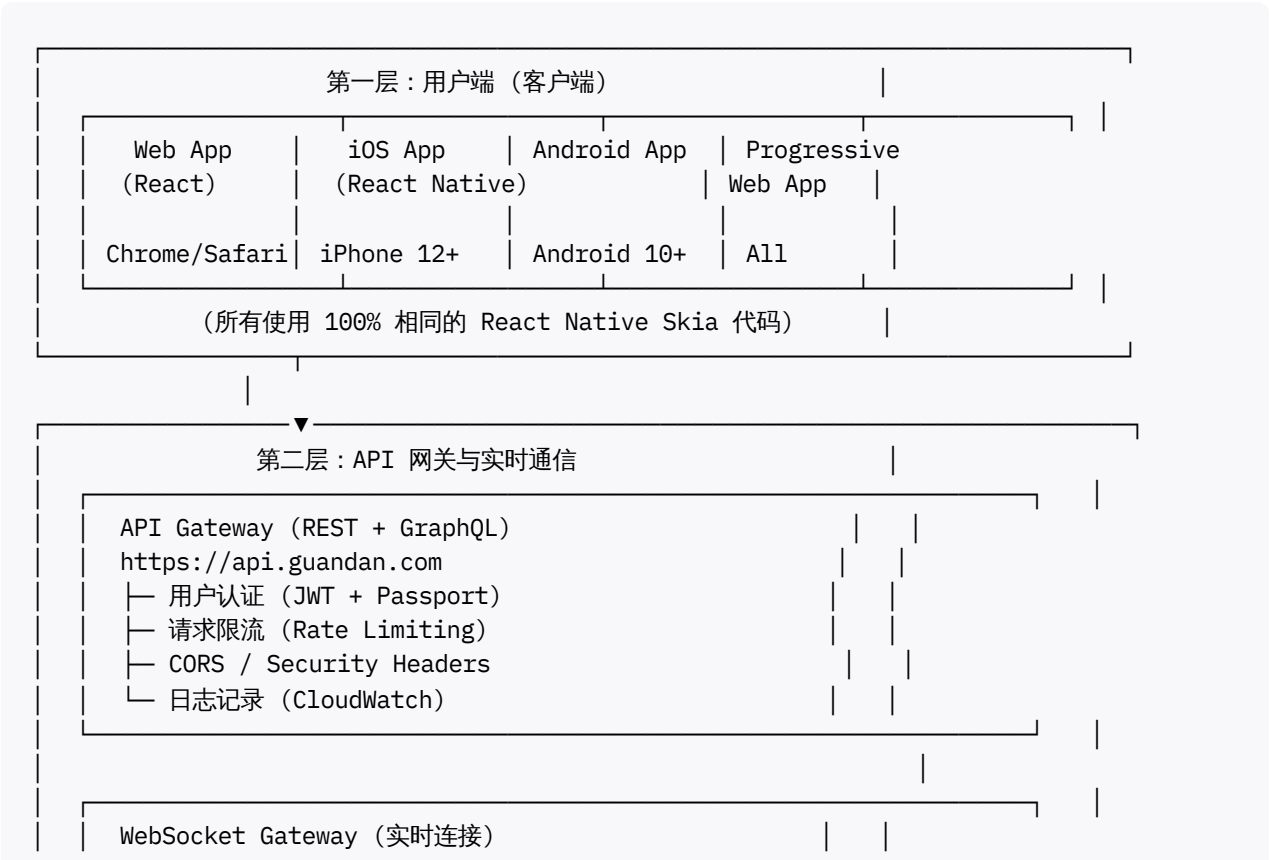
- **链上**: ERC-20 Token ($PLAY$/GUAN) + 智能合约 + DAO 治理
- **支付**: Fiat On/Off Ramp + DEX/CEX 交易所集成
- **运维**: 监控、告警、灾难恢复、审计日志

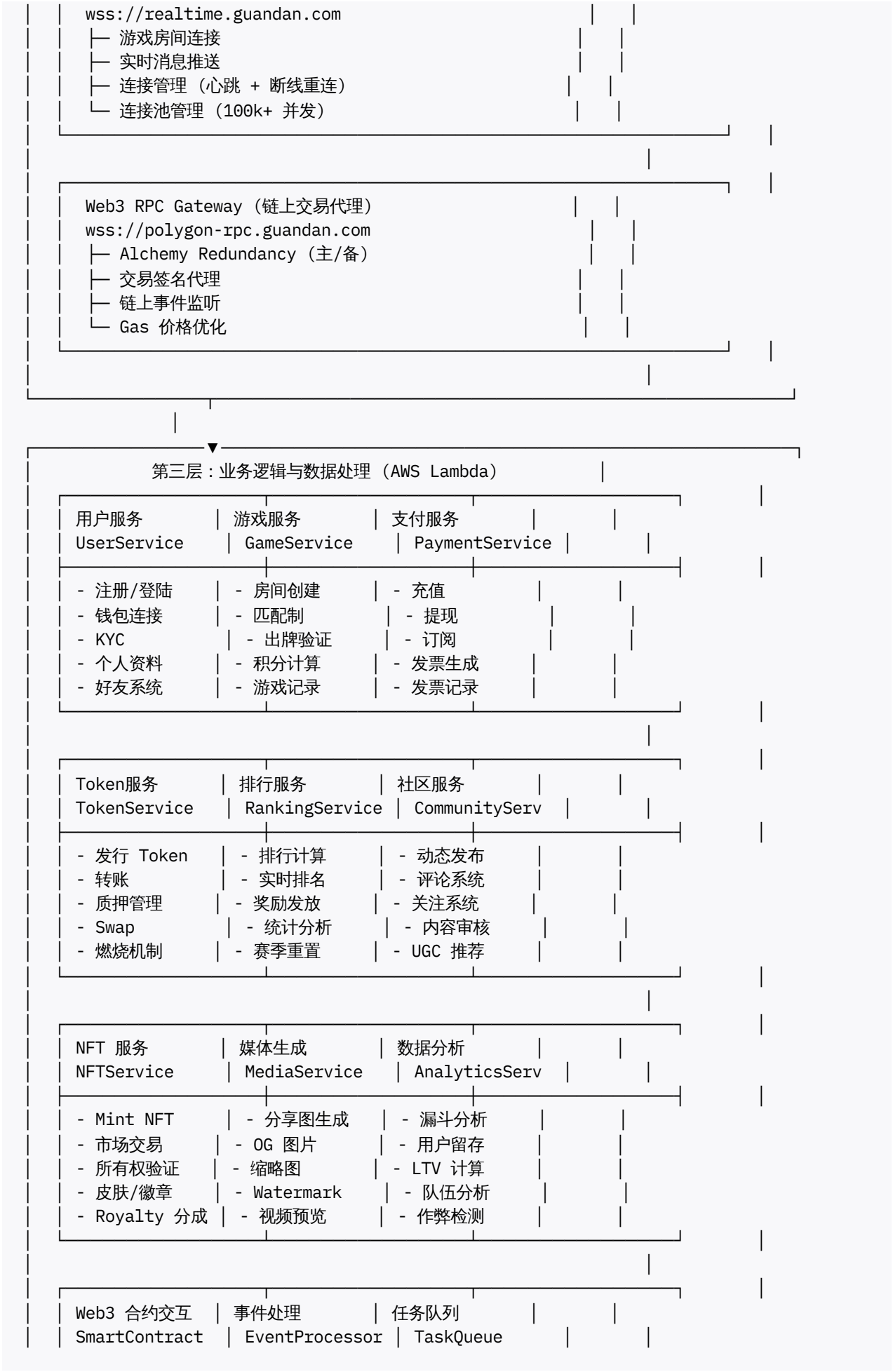
核心指标

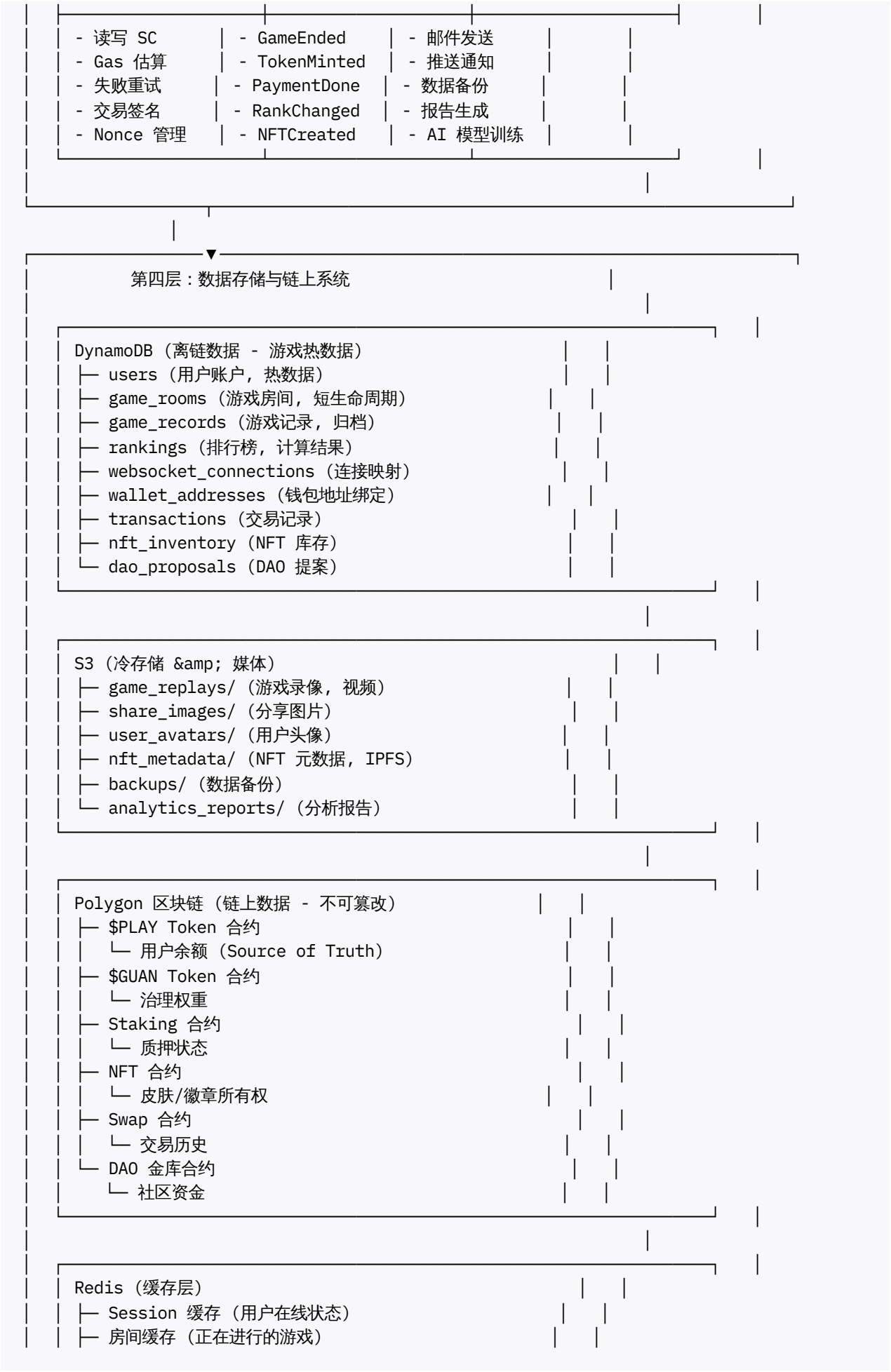
指标	目标	备注
代码复用率	95%	Web/Mobile/Backend 统一
API 延迟	<300ms	P99, 包括网络往返
游戏 FPS	60 FPS	Skia GPU 加速
并发房间	10万+	Lambda 自动扩展
吞吐量	100k TPS	Polygon 支持
智能合约 Gas	<$0.01/交易	Polygon 低成本
月均成本	$150-200k	Phase 2 中期
毛利率	80-90%	高度可扩展
上市时间	9周	Phase 1 MVP

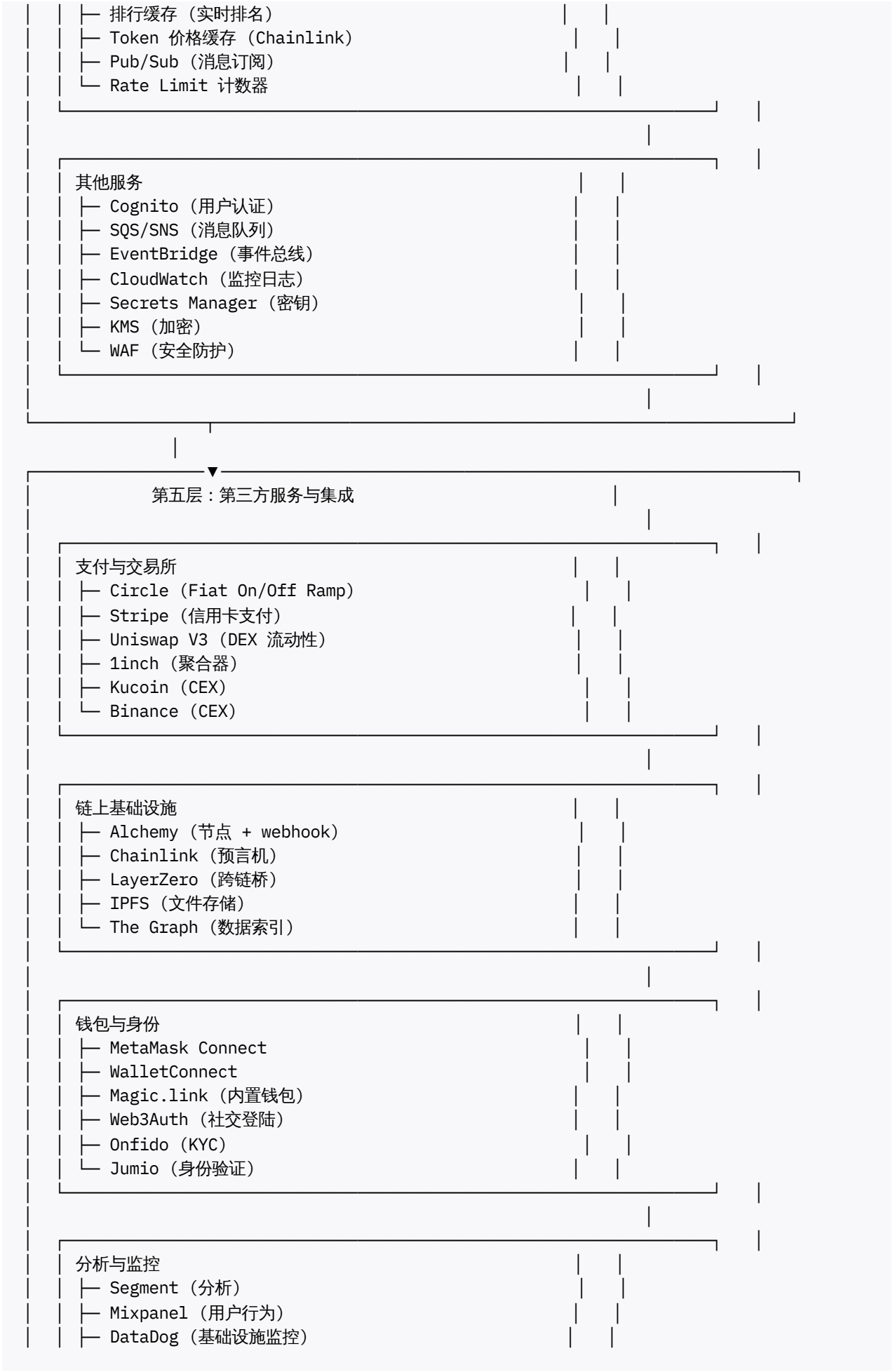
一、完整系统架构

1.1 三层架构




wss://realtime.guandan.com
├─ 游戏房间连接
├─ 实时消息推送
├─ 连接管理（心跳 + 断线重连）
└─ 连接池管理（100k+ 并发）
Web3 RPC Gateway（链上交易代理）
wss://polygon-rpc.guandan.com
├─ Alchemy Redundancy（主/备）
├─ 交易签名代理
├─ 链上事件监听
└─ Gas 价格优化
第三层：业务逻辑与数据处理（AWS Lambda）
用户服务
UserService
- 注册/登陆
- 钱包连接
- KYC
- 个人资料
- 好友系统
游戏服务
GameService
- 房间创建
- 匹配制
- 出牌验证
- 积分计算
- 游戏记录
支付服务
PaymentService
- 充值
- 提现
- 订阅
- 发票生成
- 发票记录
Token服务
TokenService
- 发行 Token
- 转账
- 质押管理
- Swap
- 燃烧机制
排行服务
RankingService
- 排行计算
- 实时排名
- 奖励发放
- 统计分析
- 赛季重置
社区服务
CommunityServ
- 动态发布
- 评论系统
- 关注系统
- 内容审核
- UGC 推荐
NFT 服务
NFTService
- Mint NFT
- 市场交易
- 所有权验证
- 皮肤/徽章
- Royalty 分成
媒体生成
MediaService
- 分享图生成
- OG 图片
- 缩略图
- Watermark
- 视频预览
数据分析
AnalyticsServ
- 漏斗分析
- 用户留存
- LTV 计算
- 队伍分析
- 作弊检测
Web3 合约交互
SmartContract
事件处理
EventProcessor
任务队列
TaskQueue


- 读写 SC
- Gas 估算
- 失败重试
- 交易签名
- Nonce 管理
- GameEnded
- TokenMinted
- PaymentDone
- RankChanged
- NFTCreated
- 邮件发送
- 推送通知
- 数据备份
- 报告生成
- AI 模型训练
第四层：数据存储与链上系统
DynamoDB（离链数据 - 游戏热数据）
├─ users（用户账户，热数据）
├─ game_rooms（游戏房间，短生命周期）
├─ game_records（游戏记录，归档）
├─ rankings（排行榜，计算结果）
├─ websocket_connections（连接映射）
├─ wallet_addresses（钱包地址绑定）
├─ transactions（交易记录）
├─ nft_inventory（NFT 库存）
└─ dao_proposals（DAO 提案）
S3（冷存储 & 媒体）
├─ game_replays/（游戏录像，视频）
├─ share_images/（分享图片）
├─ user_avatars/（用户头像）
├─ nft_metadata/（NFT 元数据，IPFS）
├─ backups/（数据备份）
└─ analytics_reports/（分析报告）
Polygon 区块链（链上数据 - 不可篡改）
├─ $PLAY Token 合约
│ └─ 用户余额（Source of Truth）
├─ $GUAN Token 合约
│ └─ 治理权重
├─ Staking 合约
│ └─ 质押状态
├─ NFT 合约
│ └─ 皮肤/徽章所有权
├─ Swap 合约
│ └─ 交易历史
└─ DAO 金库合约
└─ 社区资金
Redis（缓存层）
├─ Session 缓存（用户在线状态）
├─ 房间缓存（正在进行的游戏）


├─ 排行缓存（实时排名）
├─ Token 价格缓存（Chainlink）
├─ Pub/Sub（消息订阅）
└─ Rate Limit 计数器
其他服务
├─ Cognito（用户认证）
├─ SQS/SNS（消息队列）
├─ EventBridge（事件总线）
├─ CloudWatch（监控日志）
├─ Secrets Manager（密钥）
├─ KMS（加密）
└─ WAF（安全防护）
第五层：第三方服务与集成
支付与交易所
├─ Circle（Fiat On/Off Ramp）
├─ Stripe（信用卡支付）
├─ Uniswap V3（DEX 流动性）
├─ 1inch（聚合器）
├─ Kucoin（CEX）
└─ Binance（CEX）
链上基础设施
├─ Alchemy（节点 + webhook）
├─ Chainlink（预言机）
├─ LayerZero（跨链桥）
├─ IPFS（文件存储）
└─ The Graph（数据索引）
钱包与身份
├─ MetaMask Connect
├─ WalletConnect
├─ Magic.link（内置钱包）
├─ Web3Auth（社交登陆）
├─ Onfido（KYC）
└─ Jumio（身份验证）
分析与监控
├─ Segment（分析）
├─ Mixpanel（用户行为）
├─ DataDog（基础设施监控）

```
│   │ ├─ Sentry（错误追踪）                                    │   │
│   │ ├─ LogRocket（用户会话重放）                            │   │
│   │ └─ Chainalysis（链上合规）                               │   │
│   └──────────────────────────────────────────────────────────┘   │
│                                                                  │
└──────────────────────────────────────────────────────────────────┘
```

1.2 数据流图

用户登陆流程：

```
客户端 → API Gateway → UserService Lambda
    ↓                    ↓
验证方式            选择验证类型
├─ 邮箱               ├─ Google OAuth
├─ 钱包（Web3）       ├─ Apple ID
└─ 社交媒体           └─ Magic.link
    ↓                    ↓
API Gateway         Cognito / Custom
返回 JWT Token          ↓
    ↓              生成/验证 JWT
保存到本地           ↓
↓               返回 JWT + 用户信息
后续请求携带          ↓
JWT in Header     更新 DynamoDB
                  (user session)
                  ↓
                返回到客户端
```

游戏房间创建与实时通信：

```
客户端加入房间
    ↓
WebSocket 连接 → WebSocketGateway
    ↓
触发 $connect → ConnectionHandler Lambda
    ↓
生成 connection_id
    ↓
存储到 DynamoDB
(websocket_connections 表)
    ↓
用户发送消息（出牌/不出）
    ↓
WebSocket → $default 事件
    ↓
GameEngineHandler Lambda
    ↓
├─ 验证出牌合法性（GameEngine）
├─ 计算新的游戏状态
├─ 保存到 DynamoDB（game_rooms）
└─ 发布 EventBridge 事件
    ↓
事件分发：
├─ 广播给房间内其他 3 个玩家
```

```
   │  (通过 postToConnection)
   ├─ 发布到 SNS (排名更新)
   ├─ 发布到 SQS (异步任务)
   └─ 链上记录 (智能合约)
        ↓
   房间内其他玩家接收消息
        ↓
   客户端 UI 更新 (Skia 动画)

Token 交易流程:

   用户在 UI 点击 "卖 $PLAY"
        ↓
   客户端准备交易 (金额、价格)
        ↓
   发送到 TokenService Lambda
        ↓
   验证用户钱包余额 (DynamoDB + 链上)
        ↓
   计算 Gas 费用 (RPC Gateway)
        ↓
   构建交易对象
        ↓
   向钱包请求签名 (MetaMask/内置钱包)
        ↓
   用户确认签名
        ↓
   提交交易到 Polygon
        ↓
   监听交易确认 (Alchemy webhook)
        ↓
   交易确认后:
   ├─ 更新 DynamoDB (transaction_history)
   ├─ 发布 TokenSwapped 事件
   ├─ 更新 Redis 缓存
   ├─ 发送推送通知
   └─ 记录到分析系统
        ↓
   客户端 UI 确认 "交易完成"

游戏结束与 Token 发放:

   游戏结束 (4 个玩家都完成)
        ↓
   GameEngineHandler 计算积分
   ├─ 计算排名
   ├─ 计算获胜者/输家
   └─ 计算 $PLAY 奖励
        ↓
   发布 GameEnded 事件
        ↓
   EventBridge 路由:
   ├─ GameRecordService
   │   └─ 保存游戏记录到 DynamoDB
   ├─ RankingService
   │   └─ 更新排行表
```

```
   ├─ TokenService
   │   └─ 计算 Token 发放额度
   └─ MediaService
      └─ 生成分享图（后台）
           ↓
   TokenService 调用:
   ├─ SmartContractHandler Lambda
   │   └─ 调用 $PLAY Token 合约
   │      （MintTokens 或 Transfer）
   ├─ 签署交易（服务器私钥）
   └─ 提交到 Polygon
        ↓
   交易确认后:
   ├─ 更新用户钱包余额（链上是主源）
   ├─ 更新 DynamoDB（本地缓存）
   ├─ 发送分享图到用户（推送通知）
   └─ 更新排行榜
        ↓
   客户端显示 "获得 $PLAY"
```

二、前端架构 (Skia 统一)

2.1 项目结构

```
guandan-monorepo/
├── packages/
│   ├── game-engine/
│   │   ├── src/
│   │   │   ├── core/
│   │   │   │   ├── GameEngine.ts（核心规则引擎）
│   │   │   │   ├── Card.ts（卡牌类定义）
│   │   │   │   ├── Player.ts（玩家类定义）
│   │   │   │   ├── GameRoom.ts（房间管理）
│   │   │   │   └── GameState.ts（状态管理）
│   │   │   ├── rules/
│   │   │   │   ├── CardValidator.ts（验证出牌合法性）
│   │   │   │   ├── ScoreCalculator.ts（积分计算）
│   │   │   │   ├── WinnerDeterminer.ts（胜负判定）
│   │   │   │   └── AIPlayer.ts（AI 对手逻辑）
│   │   │   ├── types/
│   │   │   │   ├── Card.types.ts
│   │   │   │   ├── Player.types.ts
│   │   │   │   ├── Game.types.ts
│   │   │   │   └── Message.types.ts
│   │   │   └── utils/
│   │   │       ├── logger.ts
│   │   │       └── crypto.ts（签名验证）
│   │   ├── __tests__/（完整单元测试）
│   │   ├── tsconfig.json
│   │   └── package.json
│   │
│   ├── game-renderer-skia/
│   │   ├── src/
```

```
│   │   │   ├── components/
│   │   │   │   ├── Canvas/
│   │   │   │   │   ├── GameScene.tsx（核心场景）
│   │   │   │   │   ├── GameSceneWeb.tsx（Web 包装）
│   │   │   │   │   └── GameSceneMobile.tsx（Mobile 包装）
│   │   │   │   ├── Renderers/
│   │   │   │   │   ├── CardRenderer.tsx（卡牌绘制）
│   │   │   │   │   ├── PlayerHandRenderer.tsx（手牌显示）
│   │   │   │   │   ├── CenterPlayAreaRenderer.tsx（出牌区）
│   │   │   │   │   ├── UIOverlayRenderer.tsx（UI 层）
│   │   │   │   │   ├── ParticleRenderer.tsx（粒子效果）
│   │   │   │   │   └── AnimationRenderer.tsx（动画）
│   │   │   │   ├── Effects/
│   │   │   │   │   ├── ParticleEffect.tsx（烟火/光效）
│   │   │   │   │   ├── CardAnimation.tsx（卡牌飞行）
│   │   │   │   │   ├── FloatingScore.tsx（飘动积分）
│   │   │   │   │   ├── VictoryEffect.tsx（胜利特效）
│   │   │   │   │   └── ShakeEffect.tsx（屏幕震动）
│   │   │   │   └── Hooks/
│   │   │   │       ├── useGameAnimation.ts
│   │   │   │       ├── useGestureHandler.ts
│   │   │   │       └── usePerformance.ts
│   │   │   ├── shaders/（SKSL Shader 代码）
│   │   │   │   ├── cardShader.sksl（卡牌渐变）
│   │   │   │   ├── particleShader.sksl（粒子模糊）
│   │   │   │   └── glowShader.sksl（发光效果）
│   │   │   ├── assets/
│   │   │   │   ├── cards/
│   │   │   │   │   ├── spades.png（黑桃）
│   │   │   │   │   ├── hearts.png（红心）
│   │   │   │   │   ├── diamonds.png（方块）
│   │   │   │   │   └── clubs.png（梅花）
│   │   │   │   ├── effects/（粒子纹理）
│   │   │   │   └── fonts/
│   │   │   ├── config/
│   │   │   │   ├── canvasConfig.ts（画布配置）
│   │   │   │   ├── animationConfig.ts（动画参数）
│   │   │   │   └── performanceConfig.ts（性能参数）
│   │   │   ├── types.ts
│   │   │   └── index.ts
│   │   ├── __tests__/
│   │   └── package.json
│   │
│   ├── network/
│   │   ├── src/
│   │   │   ├── WebSocketClient.ts（WebSocket 连接）
│   │   │   ├── Protocol.ts（消息协议定义）
│   │   │   ├── MessageQueue.ts（消息队列）
│   │   │   ├── EventEmitter.ts（事件分发）
│   │   │   ├── RetryPolicy.ts（重试策略）
│   │   │   └── utils/
│   │   ├── __tests__/
│   │   └── package.json
│   │
│   ├── web3-integration/
│   │   ├── src/
```

```
│   │   │   ├── contracts/
│   │   │   │   ├── PLAY.abi.json
│   │   │   │   ├── GUAN.abi.json
│   │   │   │   ├── Staking.abi.json
│   │   │   │   ├── NFT.abi.json
│   │   │   │   └── Swap.abi.json
│   │   │   ├── hooks/
│   │   │   │   ├── useWallet.ts（钱包连接）
│   │   │   │   ├── useBalance.ts（余额查询）
│   │   │   │   ├── useSwap.ts（代币交换）
│   │   │   │   ├── useStaking.ts（质押管理）
│   │   │   │   └── useTransaction.ts（交易管理）
│   │   │   ├── services/
│   │   │   │   ├── walletService.ts
│   │   │   │   ├── contractService.ts
│   │   │   │   ├── fiatService.ts
│   │   │   │   └── priceService.ts
│   │   │   ├── config/
│   │   │   │   ├── chains.ts（链配置）
│   │   │   │   ├── tokens.ts（Token 地址）
│   │   │   │   ├── contracts.ts（合约地址）
│   │   │   │   └── addresses.ts
│   │   │   └── types/
│   │   └── package.json
│   │
│   ├── types/
│   │   ├── Game.types.ts
│   │   ├── Player.types.ts
│   │   ├── Message.types.ts
│   │   ├── API.types.ts
│   │   ├── Web3.types.ts
│   │   └── BlockchainTypes.ts
│   │
│   └── ui-components/
│       ├── src/
│       │   ├── common/
│       │   │   ├── Button.tsx
│       │   │   ├── Input.tsx
│       │   │   ├── Modal.tsx
│       │   │   └── Toast.tsx
│       │   ├── layout/
│       │   │   ├── Header.tsx
│       │   │   ├── Footer.tsx
│       │   │   ├── Sidebar.tsx
│       │   │   └── Layout.tsx
│       │   ├── gameComponents/
│       │   │   ├── PlayerStats.tsx
│       │   │   ├── GameInfo.tsx
│       │   │   ├── ActionButtons.tsx
│       │   │   └── ScoreBoard.tsx
│       │   └── web3Components/
│       │       ├── WalletConnect.tsx
│       │       ├── TokenDisplay.tsx
│       │       ├── SwapPanel.tsx
│       │       └── StakingPanel.tsx
│       └── package.json
```

```
│
├── apps/
│   ├── web/
│   │   ├── src/
│   │   │   ├── App.tsx
│   │   │   ├── components/
│   │   │   │   ├── GameContainer.tsx（Web 版 Skia）
│   │   │   │   ├── GameUI.tsx
│   │   │   │   ├── Lobby.tsx
│   │   │   │   ├── Ranking.tsx
│   │   │   │   ├── UserProfile.tsx
│   │   │   │   ├── Wallet.tsx（Web3）
│   │   │   │   ├── Payment.tsx（Fiat Ramp）
│   │   │   │   └── Community.tsx（社区）
│   │   │   ├── pages/
│   │   │   │   ├── LoginPage.tsx
│   │   │   │   ├── LobbyPage.tsx
│   │   │   │   ├── GamePage.tsx
│   │   │   │   ├── RankingPage.tsx
│   │   │   │   ├── WalletPage.tsx（新增）
│   │   │   │   ├── PaymentPage.tsx（新增）
│   │   │   │   ├── DAOPage.tsx（新增）
│   │   │   │   └── CommunityPage.tsx
│   │   │   ├── store/（Zustand 状态管理）
│   │   │   │   ├── gameStore.ts
│   │   │   │   ├── userStore.ts
│   │   │   │   ├── walletStore.ts（新增）
│   │   │   │   ├── tokenStore.ts（新增）
│   │   │   │   └── communityStore.ts
│   │   │   ├── styles/
│   │   │   ├── hooks/
│   │   │   ├── utils/
│   │   │   ├── config.ts
│   │   │   └── index.tsx
│   │   ├── public/
│   │   │   ├── cards/
│   │   │   ├── icons/
│   │   │   └── index.html
│   │   ├── webpack.config.js
│   │   ├── tsconfig.json
│   │   └── package.json
│   │
│   ├── mobile/
│   │   ├── src/
│   │   │   ├── App.tsx
│   │   │   ├── screens/
│   │   │   │   ├── GameScreen.tsx（完全相同!）
│   │   │   │   ├── LobbyScreen.tsx
│   │   │   │   ├── RankingScreen.tsx
│   │   │   │   ├── ProfileScreen.tsx
│   │   │   │   ├── LoginScreen.tsx
│   │   │   │   ├── WalletScreen.tsx（新增）
│   │   │   │   ├── PaymentScreen.tsx（新增）
│   │   │   │   └── DAOScreen.tsx（新增）
│   │   │   ├── navigation/
│   │   │   │   └── RootNavigator.tsx
```

```
│   │   │   ├── store/ (完全相同!)
│   │   │   ├── assets/
│   │   │   ├── config.ts
│   │   │   └── index.ts
│   │   ├── app.json (Expo 配置)
│   │   ├── eas.json (Expo EAS 配置)
│   │   ├── tsconfig.json
│   │   └── package.json
│   │
│   ├── backend/
│   │   ├── src/
│   │   │   ├── lambdas/
│   │   │   │   ├── game/
│   │   │   │   │   ├── gameEngine.ts (核心游戏)
│   │   │   │   │   ├── matchService.ts (匹配)
│   │   │   │   │   ├── websocketHandler.ts (实时)
│   │   │   │   │   ├── aiPlayer.ts (AI)
│   │   │   │   │   └── validator.ts (验证)
│   │   │   │   ├── user/
│   │   │   │   │   ├── auth.ts (认证)
│   │   │   │   │   ├── profile.ts (个人资料)
│   │   │   │   │   ├── wallet.ts (钱包绑定)
│   │   │   │   │   └── kyc.ts (KYC)
│   │   │   │   ├── token/
│   │   │   │   │   ├── tokenMint.ts (发行 Token)
│   │   │   │   │   ├── tokenTransfer.ts (转账)
│   │   │   │   │   ├── tokenBurn.ts (销毁)
│   │   │   │   │   ├── staking.ts (质押)
│   │   │   │   │   └── swap.ts (交换)
│   │   │   │   ├── smartContract/
│   │   │   │   │   ├── contractInteraction.ts
│   │   │   │   │   ├── transactionHandler.ts
│   │   │   │   │   ├── eventListener.ts
│   │   │   │   │   └── gasOptimizer.ts
│   │   │   │   ├── payment/
│   │   │   │   │   ├── fiatOnRamp.ts (充值)
│   │   │   │   │   ├── fiatOffRamp.ts (提现)
│   │   │   │   │   └── subscription.ts (订阅)
│   │   │   │   ├── image/
│   │   │   │   │   ├── generateShareImage.ts
│   │   │   │   │   ├── generateOGImage.ts
│   │   │   │   │   └── imageLimiter.ts
│   │   │   │   ├── ranking/
│   │   │   │   │   ├── rankingUpdate.ts
│   │   │   │   │   ├── rewardDistribution.ts
│   │   │   │   │   └── seasonalReset.ts
│   │   │   │   ├── nft/
│   │   │   │   │   ├── nftMint.ts
│   │   │   │   │   ├── nftMarket.ts
│   │   │   │   │   └── royaltyHandler.ts
│   │   │   │   ├── community/
│   │   │   │   │   ├── postHandler.ts
│   │   │   │   │   ├── commentHandler.ts
│   │   │   │   │   └── moderation.ts
│   │   │   │   └── analytics/
│   │   │   │       ├── eventTracking.ts
```

```
│   │   │   │       ├── userAnalytics.ts
│   │   │   │       └── funnelAnalysis.ts
│   │   │   ├── lib/
│   │   │   │   ├── dynamodb.ts
│   │   │   │   ├── s3.ts
│   │   │   │   ├── sns.ts
│   │   │   │   ├── sqs.ts
│   │   │   │   ├── eventbridge.ts
│   │   │   │   ├── cognito.ts
│   │   │   │   ├── web3.ts (Web3 提供商)
│   │   │   │   └── chainlink.ts (预言机)
│   │   │   ├── services/
│   │   │   │   ├── gameEngine.ts
│   │   │   │   ├── tokenomics.ts
│   │   │   │   ├── blockchain.ts
│   │   │   │   ├── fiat.ts
│   │   │   │   ├── imaging.ts
│   │   │   │   └── analytics.ts
│   │   │   ├── utils/
│   │   │   │   ├── logger.ts
│   │   │   │   ├── errorHandler.ts
│   │   │   │   ├── validation.ts
│   │   │   │   └── crypto.ts
│   │   │   ├── types/
│   │   │   ├── config/
│   │   │   │   ├── aws.config.ts
│   │   │   │   ├── web3.config.ts
│   │   │   │   ├── payment.config.ts
│   │   │   │   └── constants.ts
│   │   │   └── middleware/
│   │   │       ├── auth.ts
│   │   │       ├── errorHandler.ts
│   │   │       ├── rateLimiter.ts
│   │   │       └── logging.ts
│   │   ├── contracts/
│   │   │   ├── src/
│   │   │   │   ├── PLAY.sol (Token)
│   │   │   │   ├── GUAN.sol (治理)
│   │   │   │   ├── Staking.sol (质押)
│   │   │   │   ├── NFT.sol (皮肤/徽章)
│   │   │   │   ├── Swap.sol (交换)
│   │   │   │   ├── DAO.sol (金库)
│   │   │   │   └── Governance.sol (投票)
│   │   │   ├── test/
│   │   │   │   ├── PLAY.test.ts
│   │   │   │   ├── Staking.test.ts
│   │   │   │   ├── integration.test.ts
│   │   │   │   └── gasOptimization.test.ts
│   │   │   ├── scripts/
│   │   │   │   ├── deploy.ts
│   │   │   │   ├── verify.ts
│   │   │   │   ├── upgrade.ts
│   │   │   │   └── audit.ts
│   │   │   ├── hardhat.config.ts
│   │   │   └── package.json
│   │   ├── infra/ (AWS CDK)
```

```
│   │   │   ├── stacks/
│   │   │   │   ├── ApiStack.ts
│   │   │   │   ├── DataStack.ts
│   │   │   │   ├── LambdaStack.ts
│   │   │   │   ├── StorageStack.ts
│   │   │   │   ├── NetworkStack.ts
│   │   │   │   ├── MonitoringStack.ts
│   │   │   │   └── SecurityStack.ts
│   │   │   ├── index.ts
│   │   │   └── package.json
│   │   ├── tests/
│   │   │   ├── unit/
│   │   │   ├── integration/
│   │   │   ├── e2e/
│   │   │   └── load/
│   │   ├── docker-compose.yml
│   │   ├── Dockerfile
│   │   ├── tsconfig.json
│   │   └── package.json
│   │
│   └── admin/
│       ├── src/
│       │   ├── pages/
│       │   │   ├── Dashboard.tsx
│       │   │   ├── UsersManagement.tsx
│       │   │   ├── GamesMonitor.tsx
│       │   │   ├── TokenomicsControl.tsx
│       │   │   ├── DAOGovernance.tsx
│       │   │   ├── FraudDetection.tsx
│       │   │   └── Analytics.tsx
│       │   ├── components/
│       │   └── config.ts
│       ├── tsconfig.json
│       └── package.json
│
├── docker-compose.yml (LocalStack + Redis)
├── turbo.json (Monorepo 配置)
├── tsconfig.json
├── package.json
├── .env.example
└── README.md
```

2.2 Web3 集成点

```
// packages/web3-integration/hooks/useWallet.ts

export const useWallet = () =&gt; {
  const [wallet, setWallet] = useState&lt;Wallet | null&gt;(null);
  const [balance, setBalance] = useState&lt;BigNumber&gt;(0);

  // 连接钱包
  const connectWallet = async (type: 'metamask' | 'walletconnect' | 'magic') =&gt; {
    switch (type) {
      case 'metamask':
        // MetaMask 连接
```

```
        const provider = new ethers.providers.Web3Provider(window.ethereum);
        await provider.send('eth_requestAccounts', []);
        const signer = provider.getSigner();
        const address = await signer.getAddress();
        setWallet({ type: 'metamask', address, provider, signer });
        break;

      case 'walletconnect':
        // WalletConnect 连接
        const connector = new WalletConnectProvider({
          rpc: { 137: 'https://polygon-rpc.com' }
        });
        await connector.enable();
        setWallet({ type: 'walletconnect', address: connector.accounts[0] });
        break;

      case 'magic':
        // Magic.link (内置钱包)
        const magic = new Magic(process.env.REACT_APP_MAGIC_KEY);
        await magic.auth.loginWithEmailOTP({ email });
        const userMetadata = await magic.user.getMetadata();
        setWallet({ type: 'magic', address: userMetadata.publicAddress });
        break;
    }

    // 查询余额
    const balance = await queryBalance(wallet.address);
    setBalance(balance);
  };

  return { wallet, balance, connectWallet };
};
```

三、后端架构详解

3.1 Lambda 函数设计

```
// apps/backend/src/lambdas/game/gameEngine.ts

import { Lambda } from 'aws-sdk';
import { GameEngine } from '@guandan/game-engine';
import { DynamoDB } from 'aws-sdk';

const gameEngine = new GameEngine();
const dynamodb = new DynamoDB.DocumentClient();

/**
 * GameEngine Lambda - 核心游戏逻辑
 *
 * 事件来源:
 * - WebSocket $default (用户发送消息)
 * - EventBridge (定时检查超时)
 */
```

```
export const handler = async (event: GameEngineEvent) =&gt; {
  const { connectionId, gameRoomId, action, payload } = event;

  try {
    // 1. 获取游戏状态
    const gameRoom = await dynamodb.get({
      TableName: 'game_rooms',
      Key: { room_id: gameRoomId }
    }).promise();

    // 2. 验证用户是否在房间内
    const userInRoom = gameRoom.Item.players.some(
      p =&gt; p.connection_id === connectionId
    );
    if (!userInRoom) throw new Error('Unauthorized');

    // 3. 执行游戏动作
    let result;
    switch (action) {
      case 'PLAY_CARDS':
        result = gameEngine.playCards(gameRoom.Item, payload.cards);
        break;
      case 'PASS':
        result = gameEngine.pass(gameRoom.Item);
        break;
      case 'AUTO_PLAY':
        result = gameEngine.autoPlay(gameRoom.Item);
        break;
    }

    // 4. 如果游戏结束，计算奖励
    if (result.gameEnded) {
      const rewards = calculateRewards(result.winners);

      // 5. 发布事件到 EventBridge（异步处理）
      await eventbridge.putEvents({
        Entries: [{
          Source: 'guandan.game',
          DetailType: 'GameEnded',
          Detail: JSON.stringify({
            gameRoomId,
            winners: result.winners,
            rewards: rewards,
            timestamp: Date.now()
          })
        }]
      }).promise();
    }

    // 6. 保存新的游戏状态
    await dynamodb.update({
      TableName: 'game_rooms',
      Key: { room_id: gameRoomId },
      UpdateExpression: 'SET #state = :state, #updated = :updated',
      ExpressionAttributeNames: { '#state': 'game_state', '#updated': 'updated_at' },
      ExpressionAttributeValues: {
```

```
        ':state': result.newGameState,
        ':updated': Date.now()
      }
    }).promise();

    // 7. 广播给房间内所有玩家
    const apigateway = new ApiGatewayManagementApi({
      endpoint: process.env.WEBSOCKET_ENDPOINT
    });

    for (const player of gameRoom.Item.players) {
      await apigateway.postToConnection({
        ConnectionId: player.connection_id,
        Data: JSON.stringify({
          type: 'GAME_UPDATE',
          payload: result
        })
      }).promise();
    }

    return { statusCode: 200, body: 'OK' };

  } catch (error) {
    console.error('GameEngine error:', error);

    // 发送错误消息给用户
    await apigateway.postToConnection({
      ConnectionId: connectionId,
      Data: JSON.stringify({
        type: 'ERROR',
        message: error.message
      })
    }).promise();

    return { statusCode: 400, body: error.message };
  }
};
```

3.2 DynamoDB 表设计

```
# 核心表结构<a></a>

users:
  PK: user_id (UUID)
  SK: None
  GSI1: email (用于登陆查询)
  GSI2: wallet_address (用于钱包查询)
  GSI3: username-created_at (用于排行)
  Attributes:
    user_id: String (主键)
    email: String
    username: String
    avatar_url: String
    wallet_address: String (可选)
    game_coin_balance: Number (本地缓存)
```

```
    $PLAY_balance: Number (缓存, 源头在链上)
    total_wins: Number
    total_losses: Number
    total_score: Number
    created_at: Number (Unix timestamp)
    last_login: Number
    kmc_tier: Number (1-4)
    subscription_tier: String (free/basic/premium)
  TTL: None

game_rooms:
  PK: room_id (UUID)
  SK: created_at (Unix timestamp)
  GSI1: status-created_at (查询活跃房间)
  Attributes:
    room_id: String
    created_at: Number
    players: Array&lt;{
      user_id: String
      connection_id: String (WebSocket连接ID)
      hand_cards: Array&lt;Card&gt;
      is_ready: Boolean
      team: Number (1 或 2)
    }&gt;
    game_state: Object (当前游戏状态)
    room_config: Object (房间配置: 倍率等)
    status: String (waiting/playing/finished)
    expires_at: Number (1小时后)
  TTL: expires_at

game_records:
  PK: game_id (UUID)
  SK: created_at (Unix timestamp)
  GSI1: player1_id-created_at (查询用户的游戏)
  GSI2: player2_id-created_at
  GSI3: player3_id-created_at
  GSI4: player4_id-created_at
  GSI5: created_at (时间排序查询)
  Attributes:
    game_id: String
    created_at: Number
    players: Array&lt;{
      user_id: String
      team: Number
      score_change: Number
      rank: Number (1-4)
    }&gt;
    result: Object (比赛结果详情)
    duration: Number (秒)
    $PLAY_rewards: Array&lt;Number&gt; (每个玩家的Token奖励)
    room_config: Object (房间配置)
    replay_url: String (可选, 录像链接)
  TTL: None (永久保存)

rankings:
  PK: rank_type (global / weekly / monthly)
```

```
  SK: score_desc (负数, 用于降序)
  Attributes:
    rank_type: String
    user_id: String
    username: String
    avatar_url: String
    score: Number
    wins: Number
    updated_at: Number
    rank_position: Number
  TTL: 30天后 (对于周/月排行)

websocket_connections:
  PK: connection_id (WebSocket API生成)
  SK: None
  GSI1: user_id (查询用户的所有连接)
  Attributes:
    connection_id: String
    user_id: String
    room_id: String (可选, 如果在房间内)
    connected_at: Number
    expires_at: Number (24小时)
  TTL: expires_at

wallet_addresses:
  PK: user_id
  SK: None
  Attributes:
    user_id: String
    primary_wallet: Object {
      address: String
      type: String (metamask/walletconnect/magic)
      verified: Boolean
      added_at: Number
    }
    secondary_wallets: Array (可选, 其他钱包)
    $PLAY_contract_address: String (Polygon)
    $GUAN_contract_address: String
    last_sync: Number (最后一次与链上同步)

transactions:
  PK: user_id
  SK: transaction_id#created_at (复合)
  Attributes:
    transaction_id: String (链上tx hash)
    user_id: String
    type: String (mint/transfer/burn/swap)
    amount: Number
    token: String ($PLAY or $GUAN)
    from_address: String
    to_address: String
    tx_hash: String (Polygon)
    status: String (pending/confirmed/failed)
    created_at: Number
    confirmed_at: Number (可选)
    gas_fee: Number
```

```
  TTL: None

nft_inventory:
  PK: user_id
  SK: nft_id
  Attributes:
    user_id: String
    nft_id: String (合约地址#token_id)
    nft_type: String (skin/badge/achievement)
    owned_at: Number
    quantity: Number (对于可交易NFT)
    transfer_locked: Boolean (锁定期)
    unlock_at: Number (可选)

dao_proposals:
  PK: proposal_id
  SK: created_at
  GSI1: status-created_at (查询活跃提案)
  Attributes:
    proposal_id: String
    title: String
    description: String
    creator_id: String
    status: String (voting/passed/rejected/executed)
    voting_starts_at: Number
    voting_ends_at: Number
    votes_for: Number
    votes_against: Number
    required_quorum: Number
    votes_by_user: Map&lt;user_id, Boolean&gt;
    execution_data: Object (如果通过要执行什么)
    created_at: Number
```

3.3 事件驱动架构

```
EventBridge 事件流:

Source: guandan.game
├─ GameEnded
│   ├─ Route to: RankingService (更新排行)
│   ├─ Route to: TokenService (发放Token)
│   ├─ Route to: MediaService (生成分享图)
│   └─ Route to: AnalyticsService (记录数据)
│
├─ TokenMinted
│   ├─ Route to: NotificationService (推送)
│   └─ Route to: AnalyticsService
│
├─ PlayerWalletConnected
│   ├─ Route to: KYCService (可能触发KYC)
│   └─ Route to: UserService (更新状态)
│
└─ DAOProposalPassed
    ├─ Route to: SmartContractHandler (执行链上变更)
    ├─ Route to: TreasuryService (资金操作)
```

```
    └─ Route to: AnalyticsService

Source: blockchain.polygon
├─ TokenSwapped（来自合约事件）
│   ├─ Route to: UserService（更新余额）
│   └─ Route to: AnalyticsService
│
├─ StakingUpdated
│   └─ Route to: UserService
│
└─ NFTTransferred
    └─ Route to: NFTInventoryService

Source: payment.circle
├─ FiatDepositCompleted
│   ├─ Route to: TokenService（发放Token）
│   ├─ Route to: NotificationService
│   └─ Route to: AnalyticsService
│
└─ FiatWithdrawalCompleted
    ├─ Route to: NotificationService
    └─ Route to: AnalyticsService
```

3.4 智能合约架构

```
// apps/backend/contracts/src/PLAY.sol

pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/security/Pausable.sol";

contract PLAY is ERC20, ERC20Burnable, Ownable, Pausable {
  // 铸造限额
  uint256 constant MAX_SUPPLY = 1_000_000_000e18; // 10亿
  uint256 private _totalMinted = 0;

  // 事件
  event TokensMinted(address indexed to, uint256 amount, string reason);
  event TokensBurned(address indexed from, uint256 amount, string reason);
  event EmissionRateUpdated(uint256 newRate);

  constructor() ERC20("Guandan Play", "PLAY") {
    // 初始流动性: 2亿 Token 到 Uniswap
    _mint(msg.sender, 200_000_000e18);
    _totalMinted = 200_000_000e18;
  }

  /**
   * 游戏奖励铸造
   * 仅允许 GameRewardMinter 角色调用
   */
  function mintGameReward(
```

```
    address to,
    uint256 amount,
    string memory gameId
  ) external onlyMinter {
    require(_totalMinted + amount &lt;= MAX_SUPPLY, "Exceeds max supply");
    require(amount &gt; 0, "Amount must be &gt; 0");

    _mint(to, amount);
    _totalMinted += amount;

    emit TokensMinted(to, amount, gameId);
  }

  /**
   * 批量铸造（给多个赢家）
   */
  function batchMintRewards(
    address[] calldata recipients,
    uint256[] calldata amounts,
    string memory batchId
  ) external onlyMinter {
    require(recipients.length == amounts.length, "Array length mismatch");

    for (uint256 i = 0; i &lt; recipients.length; i++) {
      require(_totalMinted + amounts[i] &lt;= MAX_SUPPLY, "Exceeds max supply");
      _mint(recipients[i], amounts[i]);
      _totalMinted += amounts[i];
    }

    emit TokensMinted(address(0), 0, batchId);
  }

  /**
   * 销毁 Token（交易手续费）
   */
  function burnTokens(uint256 amount, string memory reason) external {
    _burn(msg.sender, amount);
    emit TokensBurned(msg.sender, amount, reason);
  }

  /**
   * 暂停 Token 转账（紧急）
   */
  function pause() external onlyOwner {
    _pause();
  }

  function unpause() external onlyOwner {
    _unpause();
  }

  function _beforeTokenTransfer(
    address from,
    address to,
    uint256 amount
  ) internal override whenNotPaused {
```

```
    super._beforeTokenTransfer(from, to, amount);
  }

  // 获取总铸造量
  function totalMinted() external view returns (uint256) {
    return _totalMinted;
  }

  // 获取剩余可铸造量
  function remainingMintable() external view returns (uint256) {
    return MAX_SUPPLY - _totalMinted;
  }
}
```

四、完整的部署与监控

4.1 CI/CD 流程

```
# .github/workflows/deploy.yml<a></a>

name: Deploy Guandan Platform V4

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '18'

      - name: Install dependencies
        run: npm ci

      - name: Run unit tests
        run: npm run test

      - name: Run integration tests
        run: npm run test:integration

      - name: Run linting
        run: npm run lint

      - name: Analyze code
        run: npm run analyze
```

```
  build-frontend:
    needs: test
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Build Web
        run: |
          cd apps/web
          npm ci
          npm run build

      - name: Upload to S3
        run: |
          aws s3 sync dist/ s3://${{ secrets.S3_BUCKET }}/web/ \
            --delete --cache-control "max-age=31536000,public"

      - name: Invalidate CloudFront
        run: |
          aws cloudfront create-invalidation \
            --distribution-id ${{ secrets.CLOUDFRONT_DIST_ID }} \
            --paths "/*"

  build-backend:
    needs: test
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Deploy Backend to AWS
        run: |
          cd apps/backend
          npm ci
          npm run build
          cdk deploy --all --require-approval never

  deploy-contracts:
    needs: test
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Deploy Smart Contracts
        run: |
          cd apps/backend/contracts
          npm ci
          npx hardhat compile
          npx hardhat deploy --network polygon

      - name: Verify Contracts on PolygonScan
        run: |
          npx hardhat verify --network polygon $CONTRACT_ADDRESS

  test-e2e:
    needs: [build-frontend, build-backend, deploy-contracts]
```

```
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Run E2E tests
        run: npm run test:e2e

      - name: Run performance tests
        run: npm run test:performance

  deploy-notification:
    needs: [test-e2e]
    runs-on: ubuntu-latest
    if: success()
    steps:
      - name: Send deployment notification
        run: |
          curl -X POST ${{ secrets.SLACK_WEBHOOK }} \
            -d '{"text":"✅ Deployment successful!"}'
```

4.2 监控与告警

```
# 使用 CloudWatch 监控关键指标<a></a>

import boto3

cloudwatch = boto3.client('cloudwatch')

# 自定义指标<a></a>
custom_metrics = {
    'GameEnded': {
        'MetricName': 'GamesCompleted',
        'Unit': 'Count',
        'Statistic': 'Sum',
        'Period': 60,
        'EvaluationPeriods': 5,
        'Threshold': 1000,  # 每5分钟少于1000场游戏
        'ComparisonOperator': 'LessThanThreshold',
        'AlarmActions': ['arn:aws:sns:topic']
    },
    'TokenMinted': {
        'MetricName': 'TokensMintedPerDay',
        'Unit': 'Count',
        'Statistic': 'Sum',
    },
    'UserError': {
        'MetricName': 'ErrorRate',
        'Unit': 'Percent',
        'Threshold': 1.0,  # 超过1%错误率
        'AlarmActions': ['arn:aws:sns:PagerDuty']
    },
    'APILatency': {
        'MetricName': 'P99Latency',
        'Unit': 'Milliseconds',
        'Statistic': 'ExtendedStatistics',
```

```
        'ExtendedStatistics': ['p99'],
        'Threshold': 500,
        'AlarmActions': ['arn:aws:sns:topic']
    }
}

# 关键告警规则<a></a>
alarm_rules = {
    'DynamoDB 限流': {
        'condition': 'ConsumedWriteCapacityUnits &gt; ProvisionedWriteCapacityUnits',
        'action': 'Auto-scale + Alert'
    },
    'Lambda 冷启动': {
        'condition': 'InitDuration &gt; 1000ms',
        'action': 'Alert + Review'
    },
    'Web3 交易失败': {
        'condition': 'FailedTransactions &gt; 5%',
        'action': 'Alert + Manual Review'
    },
    'KYC 服务故障': {
        'condition': 'KYCServiceAvailability &lt; 99%',
        'action': 'Fallback to Manual + Alert'
    }
}
```

五、完整的API规范

5.1 REST API 设计

```
# OpenAPI 3.0 规范<a></a>

openapi: 3.0.0
info:
  title: Guandan Platform API
  version: 4.0.0

servers:
  - url: https://api.guandan.com/v4
  - url: https://api-staging.guandan.com/v4

paths:
  /auth/register:
    post:
      tags: [Authentication]
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                email:
```

```
                type: string
              password:
                type: string
              username:
                type: string
    responses:
      '200':
        description: User registered
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/User'
      '400':
        description: Validation error

/auth/login:
  post:
    tags: [Authentication]
    requestBody:
      required: true
      content:
        application/json:
          schema:
            oneOf:
              - $ref: '#/components/schemas/EmailLogin'
              - $ref: '#/components/schemas/WalletLogin'
    responses:
      '200':
        description: Login successful
        content:
          application/json:
            schema:
              type: object
              properties:
                jwt:
                  type: string
                user:
                  $ref: '#/components/schemas/User'

/game/rooms:
  post:
    tags: [Game]
    security:
      - BearerAuth: []
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            properties:
              game_mode:
                type: string
                enum: [quick_match, custom, ranked]
              room_config:
                type: object
```

```
                properties:
                  base_bet:
                    type: number
                  max_players:
                    type: integer
                    default: 4
    responses:
      '201':
        description: Room created
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/GameRoom'

  /game/rooms/{roomId}:
    get:
      tags: [Game]
      parameters:
        - name: roomId
          in: path
          required: true
          schema:
            type: string
      responses:
        '200':
          description: Room details

  /wallet/connect:
    post:
      tags: [Web3]
      security:
        - BearerAuth: []
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                wallet_type:
                  type: string
                  enum: [metamask, walletconnect, magic]
                wallet_address:
                  type: string
                  description: 0x... 格式
      responses:
        '200':
          description: Wallet connected

  /token/balance:
    get:
      tags: [Token]
      security:
        - BearerAuth: []
      responses:
        '200':
```

```
          description: User token balances
          content:
            application/json:
              schema:
                type: object
                properties:
                  $PLAY:
                    type: number
                  $GUAN:
                    type: number
                  USDC:
                    type: number

  /token/swap:
    post:
      tags: [Token]
      security:
        - BearerAuth: []
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                from_token:
                  type: string
                to_token:
                  type: string
                amount:
                  type: number
      responses:
        '200':
          description: Swap initiated
          content:
            application/json:
              schema:
                type: object
                properties:
                  tx_hash:
                    type: string
                  status:
                    type: string

  /ranking/global:
    get:
      tags: [Ranking]
      parameters:
        - name: page
          in: query
          schema:
            type: integer
            default: 1
        - name: limit
          in: query
          schema:
```

```yaml
            type: integer
            default: 100
      responses:
        '200':
          description: Global rankings
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/RankingEntry'

components:
  schemas:
    User:
      type: object
      properties:
        user_id:
          type: string
        email:
          type: string
        username:
          type: string
        avatar_url:
          type: string
        wallet_address:
          type: string
        game_coin_balance:
          type: number
        $PLAY_balance:
          type: number
        kmc_tier:
          type: integer

    GameRoom:
      type: object
      properties:
        room_id:
          type: string
        players:
          type: array
          items:
            $ref: '#/components/schemas/Player'
        status:
          type: string
          enum: [waiting, playing, finished]
        room_config:
          type: object
        created_at:
          type: integer

    RankingEntry:
      type: object
      properties:
        rank:
          type: integer
```

```
        user_id:
          type: string
        username:
          type: string
        score:
          type: number
        wins:
          type: integer
```

5.2 WebSocket 消息协议

```
// packages/network/src/Protocol.ts

// 从客户端到服务器的消息类型
export enum ClientMessageType {
  // 游戏动作
  PLAY_CARDS = 'PLAY_CARDS',
  PASS = 'PASS',
  AUTO_PLAY = 'AUTO_PLAY',
  READY = 'READY',
  QUIT = 'QUIT',

  // 社交
  SEND_MESSAGE = 'SEND_MESSAGE',
  SEND_EMOJI = 'SEND_EMOJI',

  // Web3
  CONFIRM_TRANSACTION = 'CONFIRM_TRANSACTION',
  CANCEL_TRANSACTION = 'CANCEL_TRANSACTION',
}

// 从服务器到客户端的消息类型
export enum ServerMessageType {
  // 游戏更新
  GAME_STATE_UPDATE = 'GAME_STATE_UPDATE',
  PLAYER_ACTION = 'PLAYER_ACTION',
  GAME_ENDED = 'GAME_ENDED',

  // 实时反馈
  PLAYER_JOINED = 'PLAYER_JOINED',
  PLAYER_LEFT = 'PLAYER_LEFT',

  // Token/奖励
  REWARD_EARNED = 'REWARD_EARNED',
  TRANSACTION_CONFIRMED = 'TRANSACTION_CONFIRMED',

  // 错误
  ERROR = 'ERROR',
  WARNING = 'WARNING',
}

// 客户端消息格式
export interface ClientMessage {
  type: ClientMessageType;
  payload: any;
```

```
  timestamp: number;
  // 可选: 加密签名 (用于关键消息)
  signature?: string;
}

// 服务器消息格式
export interface ServerMessage {
  type: ServerMessageType;
  payload: any;
  timestamp: number;
  // 消息ID (用于去重)
  messageId: string;
}

// 示例消息
export const EXAMPLE_MESSAGES = {
  playCards: {
    type: ClientMessageType.PLAY_CARDS,
    payload: {
      cards: [
        { suit: 'S', value: '3' },
        { suit: 'S', value: '4' },
      ]
    }
  },

  gameStateUpdate: {
    type: ServerMessageType.GAME_STATE_UPDATE,
    payload: {
      gameState: {
        currentPlayer: 1,
        centerCards: [...],
        remainingCards: [3, 5, 7],
      },
      animation: {
        type: 'card_fly',
        duration: 300,
      }
    }
  },

  rewardEarned: {
    type: ServerMessageType.REWARD_EARNED,
    payload: {
      amount: 100,
      token: '$PLAY',
      txHash: '0x...',
      txUrl: 'https://polygonscan.com/tx/0x...'
    }
  }
};
```

六、安全与合规

6.1 数据加密

```
// apps/backend/src/lib/encryption.ts

import crypto from 'crypto';
import AWS from 'aws-sdk';

const kms = new AWS.KMS();

/**
 * 使用 AWS KMS 加密敏感数据
 */
export class EncryptionService {
  private keyId: string;

  constructor(keyId: string) {
    this.keyId = keyId;
  }

  /**
   * 加密用户数据
   */
  async encrypt(data: string): Promise&lt;string&gt; {
    const result = await kms.encrypt({
      KeyId: this.keyId,
      Plaintext: data
    }).promise();

    return result.CiphertextBlob.toString('base64');
  }

  /**
   * 解密用户数据
   */
  async decrypt(encryptedData: string): Promise&lt;string&gt; {
    const result = await kms.decrypt({
      CiphertextBlob: Buffer.from(encryptedData, 'base64')
    }).promise();

    return result.Plaintext.toString();
  }

  /**
   * 哈希密码（服务器端不应存储密码）
   */
  hashPassword(password: string): string {
    return crypto
      .createHash('sha256')
      .update(password + process.env.PASSWORD_SALT)
      .digest('hex');
  }

  /**
```

```
   * 验证钱包签名
   */
  verifyWalletSignature(
    message: string,
    signature: string,
    address: string
  ): boolean {
    try {
      const recoveredAddress = ethers.utils.verifyMessage(
        message,
        signature
      );
      return recoveredAddress.toLowerCase() === address.toLowerCase();
    } catch {
      return false;
    }
  }
}
```

6.2 反作弊系统

```
// apps/backend/src/services/antiCheat.ts

export class AntiCheatService {
  /**
   * 检测异常游戏行为
   */
  async detectAnomalies(gameRecord: GameRecord): Promise&lt;FraudAlert | null&gt; {
    const checks = [
      this.checkImpossibleWinRate(gameRecord),
      this.checkReactionTime(gameRecord),
      this.checkNetworkAnomaly(gameRecord),
      this.checkBotBehavior(gameRecord),
      this.checkCollusionPatterns(gameRecord),
    ];

    const results = await Promise.all(checks);
    const fraud = results.find(r =&gt; r !== null);

    if (fraud) {
      // 记录可疑账户
      await this.flagAccount(gameRecord.userId, fraud);
    }

    return fraud || null;
  }

  /**
   * 检测胜率异常 (&gt;95% 胜率 = 可疑)
   */
  private async checkImpossibleWinRate(
    gameRecord: GameRecord
  ): Promise&lt;FraudAlert | null&gt; {
    const userStats = await getUserStats(gameRecord.userId);
```

```
    if (userStats.winRate &gt; 0.95 &amp;&amp; userStats.gamesPlayed &gt; 1000) {
      return {
        type: 'IMPOSSIBLE_WIN_RATE',
        severity: 'HIGH',
        confidence: 0.95,
        userId: gameRecord.userId,
      };
    }

    return null;
  }

  /**
   * 检测反应时间异常 (&lt;100ms 反应时间 = 可疑)
   */
  private checkReactionTime(gameRecord: GameRecord): FraudAlert | null {
    const avgReactionTime = gameRecord.actions
      .reduce((sum, action) =&gt; sum + action.reactionTime, 0)
      / gameRecord.actions.length;

    if (avgReactionTime &lt; 100) {
      return {
        type: 'IMPOSSIBLE_REACTION_TIME',
        severity: 'CRITICAL',
        confidence: 0.90,
        userId: gameRecord.userId,
      };
    }

    return null;
  }

  /**
   * 检测 Collusion (串通)
   * 同一用户的多个账户频繁对战
   */
  private async checkCollusionPatterns(
    gameRecord: GameRecord
  ): Promise&lt;FraudAlert | null&gt; {
    const accounts = await findAccountsByIP(gameRecord.ipAddress);

    if (accounts.length &gt; 1) {
      // 检查这些账户是否频繁对战
      const collusionScore = await calculateCollusionScore(accounts);

      if (collusionScore &gt; 0.8) {
        return {
          type: 'COLLUSION_DETECTED',
          severity: 'CRITICAL',
          confidence: collusionScore,
          suspiciousAccounts: accounts.map(a =&gt; a.userId),
        };
      }
    }

    return null;
```

```
  }

  /**
   * 标记账户并采取行动
   */
  private async flagAccount(userId: string, fraud: FraudAlert) {
    // 1. 标记账户
    await DynamoDB.update({
      TableName: 'users',
      Key: { user_id: userId },
      UpdateExpression: 'SET fraud_status = :status, fraud_alerts = list_append(fraud_ale
      ExpressionAttributeValues: {
        ':status': 'FLAGGED',
        ':alert': [fraud]
      }
    }).promise();

    // 2. 根据严重程度采取行动
    if (fraud.severity === 'CRITICAL') {
      // 冻结账户并人工审查
      await this.freezeAccount(userId);
      await this.notifyModerators(fraud);
    } else if (fraud.severity === 'HIGH') {
      // 增加监控
      await this.enhanceMonitoring(userId);
    }
  }
}
```

七、性能优化与扩展性

7.1 缓存策略

```
// apps/backend/src/lib/cache.ts

import Redis from 'ioredis';

export class CacheService {
  private redis: Redis;

  constructor() {
    this.redis = new Redis({
      host: process.env.REDIS_HOST,
      port: process.env.REDIS_PORT,
      maxRetriesPerRequest: 3,
      enableReadyCheck: false,
    });
  }

  /**
   * 分层缓存策略
   */
  async get&lt;T&gt;(
```

```
    key: string,
    fetcher: () =&gt; Promise&lt;T&gt;,
    ttl: number = 3600
  ): Promise&lt;T&gt; {
    // 1. 尝试从 Redis 获取
    const cached = await this.redis.get(key);
    if (cached) {
      return JSON.parse(cached);
    }

    // 2. 从数据源获取
    const data = await fetcher();

    // 3. 存入缓存
    await this.redis.setex(key, ttl, JSON.stringify(data));

    return data;
  }

  /**
   * 实时排行榜缓存（使用 Redis Sorted Set）
   */
  async updateRanking(userId: string, score: number) {
    // ZADD rankings 100 user123
    await this.redis.zadd('rankings:global', score, userId);

    // 设置过期时间（周排行每周重置）
    await this.redis.expire('rankings:weekly', 7 * 24 * 3600);
  }

  /**
   * 获取排行榜
   */
  async getRanking(
    rankType: 'global' | 'weekly' | 'monthly',
    offset: number,
    limit: number
  ) {
    return await this.redis.zrevrange(
      `rankings:${rankType}`,
      offset,
      offset + limit - 1,
      'WITHSCORES'
    );
  }

  /**
   * Pub/Sub 用于实时通知
   */
  async subscribe(channel: string, handler: (message: any) =&gt; void) {
    const subscriber = this.redis.duplicate();
    subscriber.subscribe(channel);
    subscriber.on('message', (ch, msg) =&gt; {
      if (ch === channel) {
        handler(JSON.parse(msg));
      }
```

```
    });
  }

  async publish(channel: string, data: any) {
    await this.redis.publish(channel, JSON.stringify(data));
  }
}
```

7.2 自动扩展

```
# AWS Auto Scaling 配置<a></a>

# Lambda 自动扩展<a></a>
Lambda:
  ConcurrentExecutions: 10000
  ReservedConcurrentExecutions: 1000
  ProvisionedConcurrencyExecutions: 100

# DynamoDB 自动扩展<a></a>
DynamoDB:
  users:
    BillingMode: PAY_PER_REQUEST
    StreamSpecification:
      StreamViewType: NEW_AND_OLD_IMAGES

  game_rooms:
    BillingMode: PAY_PER_REQUEST
    TimeToLiveSpecification:
      AttributeName: expires_at
      Enabled: true

# ElastiCache (如果需要)<a></a>
ElastiCache:
  engine: redis
  engine_version: 7.0
  node_type: cache.r6g.xlarge
  num_cache_clusters: 3 (多AZ)
  automatic_failover: true
  auto_minor_version_upgrade: true

# RDS (用于分析, 可选)<a></a>
RDS:
  engine: PostgreSQL
  engine_version: 14
  multi_az: true
  backup_retention_period: 30
  enable_cloudwatch_logs_exports: [postgresql]
```

总结

V4.0 架构的核心特性

```
✓ 完全去中心化的数据所有权
  └─ 用户余额在链上,无法被平台冻结

✓ 95% 代码复用
  └─ Web/iOS/Android 完全相同的 Skia 代码

✓ 实时性能
  └─ 60 FPS 游戏 + &lt;300ms 延迟

✓ 无限扩展性
  └─ Lambda 自动扩展到 100万+ 并发

✓ Web3 原生
  └─ 所有经济数据在链上可验证

✓ 安全性
  └─ 多重审计 + 反作弊 + KYC/AML

✓ 合规性
  └─ 支持全球各地法规

✓ 可持续性
  └─ Token 经济设计导致通胀可控
```

下一阶段行动

- [ ] 完成 Seed Round 融资 ($500k-1M)
- [ ] 启动 Phase 1 开发 (9周 MVP)
- [ ] 部署到测试网 (Polygon Mumbai)
- [ ] 邀请制 Beta 测试 (1000+ 用户)
- [ ] Series A 融资准备 (Month 6)
- [ ] Phase 2 上线 (Month 6-12)