GEM Backend Comparison Report

GEM Backend Comparison Report

Date: 2026-02-11 Analyst: Claude Opus 4.6 Codebase A: `/Users/howardli/Downloads/gema-backend-main/app/` (CTO-approved) Codebase B: `/Users/howardli/Downloads/gem-platform/backend/app/` (Sprint 1 新开发)

Executive Summary

Metric	Codebase A (gema-backend-main)	Codebase B (gem-platform)
Total Lines	~3,140	~13,130
Python Files	47	108
Data Models	12 tables	21+ tables
API Endpoints	~12	~55
架构成熟度	MVP (基础抽卡)	完整产品 (盲盒+市场+回购)
认证系统	Solana Wallet Only	Wallet + Email OTP + Twitter
测试覆盖	1 test file	0 test files
安全特性	基础	Rate limiting + 审计日志 + 全局异常处理

关键发现: – Codebase B 是 A 的**全面扩展版本**，规模约 4 倍 – B 实现了 PRD v0.1.3 的 80%+ 功能，A 只完成了抽卡核心 – B 的架构设计更成熟，但引入了更多复杂度 – **两者不兼容**：数据模型、API 接口、业务逻辑都有重大差异

---

1. 架构对比

1.1 目录结构

目录	Codebase A	Codebase B	说明
`api/`	✅ 2 routers	✅ 14 routers	B 扩展了 12 个新模块
`models/`	✅ 4 models	✅ 14 models	B 新增 10 个表
`services/`	✅ 9 services	✅ 19 services	B 增加了复杂业务逻辑层
`db/`	✅ (简单 DAO)	✅ (完整 Repository 模式)	B 采用了更规范的数据访问层
`schemas/`	✅ Pydantic	✅ Pydantic	两者都用了 schema 分层
`plib/`	✅ 工具库	✅ 工具库 (复制)	B 复制了 A 的工具库
`ext_service/`	✅ Telegram	❌ 无	B 未实现 Telegram 集成
`test/`	✅ 1 test	❌ 无	**B 缺少测试**

1.2 app.py 入口对比

特性	Codebase A	Codebase B
框架	FastAPI	FastAPI
CORS	仅 DEV 环境 `*`	DEV: `*`, PROD: 配置的 frontend_url
Rate Limiting	❌ 无	✅ slowapi (100 req/min on /health)
全局异常处理	❌ 无	✅ 500 错误在 PROD 隐藏细节
健康检查	❌ 无	✅ `/health` 端点，返回已加载路由
Middleware	仅 CORS	CORS + Rate Limiter
Router 动态加载	❌ 硬编码	✅ `_try_import()` 优雅降级

特性	Codebase A	Codebase B
日志系统	手动 `get_logger()`	`setup_logging()` 统一配置

结论: B 的入口更成熟，生产就绪程度更高。

1.3 config.py 对比

配置项	Codebase A	Codebase B
数据库	✅ MySQL (db_host/user/passwd/name)	✅ MySQL (同)
Redis	✅ (host/port/db)	✅ (同)
JWT	✅ secret, id_exp	✅ secret, id_exp, jwt_algorithm
Twitter OAuth	✅ id/secret/redirect_url/api_token	❌ 无
Solana	✅ sol_key, sol_wallet	✅ sol_key, sol_api_url
Stripe	❌ 无	✅ stripe_secret_key, stripe_webhook_secret
前端 URL	❌ 无	✅ frontend_url (CORS 配置用)
Mint	✅ mint_url	✅ mint_url, mint_key
默认值	❌ 无	✅ 有默认值 (如 db_host="localhost")
生产检查	❌ 无	✅ 检查 secret 是否改

结论: B 配置更完善，支持 Stripe 支付，但**丢失了 Twitter OAuth 配置**（需补回）。

---

2. 数据模型对比

2.1 表结构全景

表名	Codebase A	Codebase B	功能
users	✅	✅	用户主表 (字段差异大)
balances	✅	❌	A 独有，积分+推荐数

表名	Codebase A	Codebase B	功能
reward_records	✅	❌	A 独有，奖励历史
referral_relationships	✅	❌	A 独有，推荐关系
referral_reward	❌	✅	B 独有，推荐奖励 (类似 A 的 reward_records)
orders	✅	❌	A 的订单表 (含物流)
redemption_order	❌	✅	B 的订单表 (实物兑换)
redemption_order_item	❌	✅	B 的订单明细
shipping_address (A)	✅	❌	A 的地址表 (1:1 order)
shipping_address (B)	❌	✅	B 的地址表 (多地址)
shipping_tracks	✅	❌	A 独有，物流追踪
payments	✅	❌	A 的支付记录
nfts (A)	✅	❌	A 的 NFT 表
nft (B)	❌	✅	B 的 NFT 表 (字段不同)
nft_collection_meta	❌	✅	B 的 NFT 集合元数据
nft_sequence	❌	✅	B 的 NFT 序号管理
nft_payment	❌	✅	B 的 NFT 支付记录
products	✅	❌	A 的商品表 (盲盒)
nft_categories	✅	❌	A 的 NFT 分类

表名	Codebase A	Codebase B	功能
unpack_strategies	✅	❌	A 的开盒策略(树形概率)
unpack_probabilities	✅	❌	A 的概率配置
shipping_fees	✅	❌	A 的运费表
pack	❌	✅	B 的盲盒表
pack_drop_table	❌	✅	B 的掉率表
pack_opening	❌	✅	B 的开盒记录
pack_version	❌	✅	B 的盲盒版本
user_vault	❌	✅	B 的用户仓库(开盒后物品)
market_listings	❌	✅	B 的市场挂单
market_offers	❌	✅	B 的市场报价
buyback_requests	❌	✅	B 的回购请求
wallet	❌	✅	B 的钱包余额
deposits	❌	✅	B 的充值记录(Stripe/USDC)
wallet_ledger	❌	✅	B 的钱包账本(审计)
admin_audit_log	❌	✅	B 的管理员操作审计
community	❌	✅	B 的社区表 (未用)

总计: A = 12 表，B = 21 表。B 新增 14 表，共享 7 表但字段不同。

2.2 users 表字段对比

字段	Codebase A	Codebase B	说明
`id`	String(255)	String(36) UUID	**类型不同**
`sol_address`	✅ unique, index	❌	A 独有
`wallet_address`	❌	✅ unique, index	B 独有 (更通用)

字段	Codebase A	Codebase B	说明
`email`	✅ unique, index	✅ unique, index	**都有**
`twitter_id`	✅ index	❌	A 独有
`twitter_handle`	❌	✅	B 独有
`twitter_name`	✅	❌	A 独有
`avatar_seed`	❌	✅	B 独有 (头像生成种子)
`credit_balance`	❌	✅ DECIMAL(20,6)	B 独有 (货币系统)
`referred_by`	❌	✅ index	B 独有 (推荐人 user_id)
`referral_code`	✅ index	❌	A 独有 (推荐码)
`wallet_version`	❌	✅ Integer	B 独有 (升级追踪)
`role`	❌	✅ String(20)	B 独有 (admin/user)
`pity_counter_legendary`	❌	✅ Integer	B 独有 (保底机制)
`is_active`	❌	✅ Boolean	B 独有 (封禁)
`last_login_at`	❌	✅ DateTime	B 独有
`created_at`	✅ DateTime	✅ DateTime	**都有**
`updated_at`	✅ DateTime	✅ DateTime	**都有**

结论: 两者 users 表完全不兼容。B 移除了 A 的关系 (Balance, RewardRecord, Nft, Order)，引入了内部字段。

2.3 盲盒/抽卡系统对比

Codebase A (products + unpack_strategies)

- **products** 表：存盲盒商品（价格、数量、图标、奖励积分）
- **unpack_strategies** 表：抽卡策略（树形结构）
- **unpack_probabilities** 表：概率配置（支持多层嵌套）
  - `next_strategy_id` OR `nft_category_id`（互斥）
  - `probability` 字段 (1000 = 100%)
- **nft_categories** 表：最终抽到的 NFT 类型
- **设计**: 支持**多轮抽卡**（先抽稀有度，再抽具体 NFT）

Codebase B (pack + pack_drop_table)

- pack 表：盲盒元数据（series_id, max_supply, price, status）
- pack_drop_table 表：稀有度掉率（COMMON/RARE/EPIC/LEGENDARY）
  - `drop_rate` DECIMAL(5,2) (0–100)
- pack_opening 表：开盒记录（user_id, quantity, status, payment_id）
- user_vault 表：开盒后物品（nft_id, rarity, fmv, status）
- 设计: 单轮抽卡（直接抽稀有度，再从 nft 表里选同稀有度的 NFT）

差异: – A 支持树形多层概率，B 是扁平稀有度表 – A 的 NFT 通过 `nft_categories` 预定义，B 的 NFT 通过 `nft` 表动态管理 – B 引入了 `user_vault` 概念（开盒后物品存在仓库，可交易/兑换/回购） – 不兼容: 需要重写抽卡引擎

2.4 订单系统对比

Codebase A (orders)

- orders 表：
  - `product_id` (FK to products)
  - `status`: NEW, PAID, CONFIRMED, EXPIRED, REFUNDED, SHIPPED
  - `order_type`: PRODUCT, SHIPPING
  - 关系: `payment`, `shipping_address`, `shipping_track`
- shipping_address 表：1:1 关系（`ForeignKey orders.id` 作为主键）
- shipping_tracks 表：物流追踪（carrier, tracking_no, seq）
- payments 表：支付记录（transaction_hash, from_address, amount）

Codebase B (redemption_order)

- redemption_order 表：
  - 用途：实物兑换订单（NFT → 钻石）
  - `status`: PENDING, CONFIRMED, PROCESSING, SHIPPED, DELIVERED, CANCELLED, CUSTOMS_HOLD, RETURNED
  - `subtotal_amount`, `shipping_fee`, `total_amount`
  - `shipping_quote_id`, `shipping_quote_expires_at` （物流报价）
  - `tracking_number`, `carrier`
- redemption_order_item 表：订单明细（支持多件 NFT）
  - `vault_item_id`, `nft_id`, `status`
  - `item_tracking_number`, `item_carrier` （单件追踪）
- shipping_address 表：用户地址簿（多地址， `user_id` + `id` 主键）
  - `is_validated`, `validation_provider` (Shippo 验证)
  - 支持多国家（`shipping_country_code`）

差异: – A 的订单 = 购买盲盒，B 的订单 = 兑换实物钻石 – A 一单一件，B 一单多件（明细表） – B 支持**动态物流报价**（quote_expires_at），A 只记录固定运费 – B 的地址系统更完善（多地址 + 验证） – **不兼容**: 业务逻辑完全不同

---

3. API 端点对比

3.1 端点数量

模块	Codebase A	Codebase B
用户/认证	~5	~10
订单	~3	~5
商品/盲盒	~2	~8
NFT	~2	~5
Marketplace	❌	~6
Buyback	❌	~4
Wallet	❌	~5
Referral	❌	~3
Rank	❌	~2
Admin	❌	~10
Webhooks	❌	~2
总计	**~12**	**~55**

3.2 用户/认证端点对比

端点	Codebase A	Codebase B	说明
GET /user/sign-in	✅	❌	A: 生成签名挑战 (Solana)
POST /user/verify	✅	❌	A: 验证签名 + 返回 JWT
POST /auth/wallet/challenge	❌	✅	B: 生成签名挑战 (通用)
POST /auth/wallet/verify	❌	✅	B: 验证签名 + 返回 JWT
POST /auth/email/otp/send	❌	✅	B: 发送邮箱 OTP

端点	Codebase A	Codebase B	说明
POST /auth/email/otp/verify	❌	✅	B: 验证 OTP + 返回 JWT
POST /auth/twitter/bind	❌	✅	B: 绑定 Twitter
POST /auth/refresh	❌	✅	B: 刷新 Token
GET /user/info	✅	❌	A: 获取用户信息
GET /users/me	❌	✅	B: 获取我的信息
PATCH /users/me	❌	✅	B: 更新我的信息
POST /user/twitter-oauth	✅	❌	A: Twitter OAuth (已实现)
POST /user/email	✅	❌	A: 更新邮箱

差异: – B 的认证系统**更完整**（Email OTP + Refresh Token + 速率限制） – A 的 Twitter OAuth **已实现**，B 只有绑定接口（未实现 OAuth 流程） – A 的签名验证是 Solana 专用，B 是通用的（支持多链） – **API 路径不兼容**: `/user` vs `/users`，`/auth` 模块

3.3 订单端点对比

端点	Codebase A	Codebase B	说明
GET /user/orders	✅	❌	A: 获取我的订单 (购买盲盒)
POST /user/orders	✅	❌	A: 创建订单
PUT /user/orders/{id}	✅	❌	A: 更新物流地址
POST /user/orders/{id}	✅	❌	A: 支付订单 (tx_hash)
POST /order/create-redemption	❌	✅	B: 创建兑换订单
GET /order/redemptions	❌	✅	B: 获取我的兑换订单
POST /order/redemptions/{id}/confirm	❌	✅	B: 确认订单
GET /order/redemptions/{id}	❌	✅	B: 获取订单详情
POST /order/shipping/quote	❌	✅	B: 获取物流报价

差异: – A 的订单 = 购买盲盒，B 的订单 = 兑换实物 – B 引入了**动态物流报价**机制 – **完全不兼容**: 业务流程不同

3.4 盲盒/Pack 端点对比

端点	Codebase A	Codebase B	说明
GET /product	✅	❌	A: 列出商品
POST /product/open	✅	❌	A: 开盒 (lottery)
GET /packs	❌	✅	B: 列出盲盒
GET /packs/{id}	❌	✅	B: 盲盒详情 + 掉率
POST /packs/{id}/purchase	❌	✅	B: 购买盲盒 (创建 opening)
POST /packs/openings/{id}/confirm	❌	✅	B: 确认支付 (tx_hash)
POST /packs/openings/{id}/open	❌	✅	B: 执行开盒 (返回 NFT)
GET /packs/openings	❌	✅	B: 我的开盒记录

差异: – A 的开盒是**同步的**（POST /product/open 直接返回结果） – B 的开盒是**三步流程**（购买 → 支付确认 → 开盒） – B 支持**批量购买**（quantity 参数） – **API 设计不兼容**

3.5 Marketplace (B 独有)

端点	功能
GET /market/listings	列出市场挂单 (分页 + 筛选)
POST /market/vault/{vault_item_id}/list	挂单卖 NFT
POST /market/listings/{id}/delist	取消挂单
POST /market/listings/{id}/offer	出价
POST /market/offers/{id}/accept	接受出价
POST /market/offers/{id}/reject	拒绝出价

3.6 Buyback (B 独有)

端点	功能
POST /buyback/request	提交回购申请
GET /buyback/requests	我的回购记录
POST /buyback/requests/{id}/cancel	取消回购
GET /buyback/eligible-items	可回购的物品

3.7 Wallet (B 独有)

端点	功能
GET /wallet/balance	我的余额
POST /wallet/deposit/stripe	Stripe 充值
POST /wallet/deposit/usdc	USDC 充值
GET /wallet/deposits	充值记录
GET /wallet/ledger	账本记录

3.8 Admin (B 独有)

端点	功能
POST /admin/pack	创建盲盒
POST /admin/nft	创建 NFT
GET /admin/audit-logs	审计日志
PATCH /admin/users/{id}	修改用户
GET /admin/stats	数据统计
等 10+ 管理端点	

结论: B 的 API 是 A 的**全面扩展**，覆盖了 PRD 的主要功能。但**完全不兼容**。

---

4. Auth 系统对比

4.1 认证方式

特性	Codebase A	Codebase B
Solana Wallet	✅ ED25519 签名验证	✅ 通用 Web3 签名验证
Email OTP	❌	✅ 6 位数 OTP，存 Redis
Twitter OAuth	✅ 已实现	⚠️ 只有绑定接口，OAuth 流程缺失
JWT Token	✅ HS256	✅ HS256
Refresh Token	❌	✅ 支持 Token 刷新
Session	✅ Redis (SessionData)	✅ Redis (LoginChallenge/OTP)
Rate Limiting	❌	✅ slowapi (20 req/min)

4.2 Token 管理

Codebase A

- `TokenService.gen_token(user)`: 生成 JWT
  - Payload: `{id: user.id, sol_address: user.sol_address}`
  - 有效期: `settings.id_exp`（默认 1 天）
- `TokenService.parse_token(token)`: 解析 JWT
- **无 Refresh Token**

Codebase B

- `AuthService.issue_wallet_challenge()`: 生成挑战 (nonce + message)
  - 存 Redis: `login_challenge:{nonce}`（TTL 5 分钟）
- `AuthService.verify_wallet_signature()`: 验证签名 + 创建用户 + 返回 Token
  - 返回: `{access_token, refresh_token, expires_in}`
- `AuthService.refresh_access_token()`: 刷新 Token
  - 解析 refresh_token → 生成新的 access_token + refresh_token
- `AuthService.send_email_otp()`: 发送 OTP
  - 存 Redis: `email_otp:{email}`（TTL 10 分钟）
- `AuthService.verify_email_otp()`: 验证 OTP + 创建/绑定用户
- `AuthService.bind_twitter()`: Twitter 绑定
  - **OAuth 流程未实现** (TODO)

差异: – B 的 Token 系统更完善（Refresh Token + 过期时间） – B 支持**多种登录方式**（Wallet + Email），A 只有 Wallet – B 的 Twitter 功能**不完整**（只有绑定，没有 OAuth 流程）

4.3 安全特性

特性	Codebase A	Codebase B
Rate Limiting	❌	✅ slowapi (API 级别)
Session 过期	❌	✅ Redis TTL (5–10 分钟)
IP/User-Agent 记录	❌	✅ (用于审计)
JWT Secret 检查	❌	✅ 生产环境必须修改
全局异常处理	❌	✅ 500 错误隐藏细节 (非 DEV)
HTTPS 强制	❌	❌ (TODO)
CSRF 保护	❌	❌ (JWT 天然防护)

结论: B 的安全性**远高于** A。

---

5. 业务逻辑对比

5.1 盲盒抽卡引擎

Codebase A (LotteryService)

```python
# 文件: app/services/lottery.py
def lottery(strategy_id: int) -> NftCategory:
    """
    树形多层抽卡:
    1. 从 unpack_strategies 找到当前策略
    2. 加载 probabilities (可能有 next_strategy_id 或 nft_category_id)
    3. 按概率随机选一个
    4. 如果是 next_strategy_id，递归调用
    5. 如果是 nft_category_id，返回 NFT 类型
    """
    probabilities = get_probabilities(strategy_id)
    total_prob = sum(p.probability for p in probabilities)
    rand = random.randint(1, total_prob)

    cumulative = 0
    for p in probabilities:
        cumulative += p.probability
        if rand <= cumulative:
            if p.next_strategy_id:
                return lottery(p.next_strategy_id)  # 递归
            else:
                return p.nft_category
```

特点: – 支持**树形概率**（可多层嵌套） – 灵活性高，但配置复杂 – 有**单元测试**（`test_lottery_service.py`）

Codebase B (PackEngineService)

```python
# 文件: app/services/pack_engine.py
def open_pack(pack_id: str, quantity: int) -> List[Nft]:
    """
    扁平抽卡:
    1. 从 pack_drop_table 加载稀有度掉率
    2. 按 items_per_pack 抽 N 次稀有度
    3. 检查保底机制 (pity_counter_legendary)
    4. 从 nft 表随机选同稀有度的 NFT
    5. 创建 user_vault 记录
    """
    drop_table = get_drop_table(pack_id)
    results = []

    for _ in range(pack.items_per_pack * quantity):
        rarity = roll_rarity(drop_table)

        # 保底检查 (E30)
        if user.pity_counter_legendary >= 100:
            rarity = Rarity.LEGENDARY
            user.pity_counter_legendary = 0

        nft = select_random_nft(rarity)
        results.append(nft)

        # 创建 vault 记录
        vault_item = UserVault(nft_id=nft.id, rarity=rarity, ...)
        db.add(vault_item)

    return results
```

特点: – **扁平概率**（只抽稀有度，不支持嵌套） – 支持**保底机制**（100 次必出传说） – 抽卡后物品进 `user_vault`（可交易） – **无测试**

差异: – A 的抽卡引擎更灵活（树形），B 的更简单（扁平） – B 引入了保底机制（PRD E30） – B 的 NFT 是动态的（从 nft 表随机选），A 的是预定义的（nft_categories） – **不兼容**: 需要重写

5.2 NFT 生命周期

Codebase A

```
1. 管理员创建 nft_categories (预定义 NFT 类型)
2. 用户购买 product (盲盒)
3. 创建 order (status: NEW)
4. 用户支付 (POST /user/orders/{id})
5. order.status → PAID
6. 后台任务验证链上支付
7. order.status → CONFIRMED
8. 调用 lottery() 抽卡
9. 创建 nft 记录 (status: UNSOLD, owner: user_id)
10. 调用 Solana mint API
11. nft.status → SOLD, minted_at 更新
12. 用户申请兑换实物
13. order.status → SHIPPED
14. 物流追踪 (shipping_tracks)
```

Codebase B

```
1. 管理员创建 pack (series_id, drop_table)
2. 管理员创建 nft 记录 (预生成 NFT 池)
3. 用户购买 pack (POST /packs/{id}/purchase)
4. 创建 pack_opening (status: PENDING)
5. 用户支付 (POST /packs/openings/{id}/confirm)
6. pack_opening.status → PAID
7. 用户开盒 (POST /packs/openings/{id}/open)
8. 调用 pack_engine.open_pack()
9. 创建 user_vault 记录 (status: VAULTED)
10. pack_opening.status → OPENED
11. 用户可选: 上架交易 (market_listings), 回购 (buyback_requests), 兑换实物
(redemption_order)
12. 兑换流程: 创建 redemption_order → shipping_quote → 确认 → SHIPPED → DELIVERED
```

差异: – A 的 NFT 是**开盒时铸造**的（mint on demand） – B 的 NFT 是**预生成的**（mint 由管理员提前完成） – B 引入了 `user_vault` 中间状态（开盒后的物品可以交易/回购/兑换） – **流程完全不同**

5.3 推荐系统

Codebase A

- `referral_relationships` 表：记录推荐关系（referee → referrer）
- `reward_records` 表：记录奖励历史（event: REFEREE/REFERRAL/PURCHASE/TWITTER_AUTH）
- `Balance` 表：存积分 (points) 和推荐数 (referrals)
- **逻辑**: 推荐人注册时记录关系，推荐人获得奖励积分

Codebase B

- `referral_reward` 表：记录推荐奖励（reward_type: SIGNUP, FIRST_ORDER）
- `User.referred_by` 字段：记录推荐人 user_id
- **逻辑**: 推荐人注册时记录 `referred_by`，推荐人获得 credit_balance（可消费的余额）

差异: – A 使用**积分系统**（points），B 使用**货币系统**（credit_balance） – A 的奖励记录更详细（event 类型），B 只记录 SIGNUP/FIRST_ORDER – **数据结构不兼容**

5.4 支付系统

Codebase A

- **仅 Solana USDC**
- 流程:
  1. 创建 order
  2. 前端调用 Phantom 钱包转账
  3. 用户提交 tx_hash
  4. 后台任务验证链上交易
  5. 确认后更新 order.status → PAID
- `payments` 表：记录交易 hash, from_address, amount

Codebase B

- **多支付方式**:
  - Stripe (信用卡)
  - Solana USDC
  - 跨链稳定币（未实现）
- 流程:
  1. 用户充值 → wallet.balance 增加
  2. 购买盲盒/创建订单 → 扣 balance
  3. 回购/退款 → 加 balance
- `deposits` 表：记录充值（deposit_type: CREDIT_CARD, SOLANA_USDC, CROSS_CHAIN_STABLE）
- `wallet_ledger` 表：审计日志（所有余额变动）
- **Stripe Webhook**: 监听支付成功事件

差异: – A 是**直接支付**（每笔订单单独付款） – B 是**钱包系统**（先充值，后消费） – B 支持**多支付渠道**，A 只有 Solana – **架构完全不同**

---

6. 外部服务对比

服务	Codebase A	Codebase B	说明
Solana	✅ web3_sol.py	✅ web3_sol.py (复制)	签名验证 + 转账验证
TON	✅ web3_ton.py	✅ web3_ton.py (复制)	TON 链支持
Twitter OAuth	✅ oauth.py	✅ oauth.py (复制, 未用)	A 已集成，B 未完成
SendGrid	✅ sendmail.py	✅ sendmail.py (复制)	邮件发送
Shippo	✅ address_api.py	✅ address_api.py (复制)	地址验证 + 物流报价
Stripe	❌	✅	信用卡支付 + Webhook
Telegram Bot	✅ ext_service/tg.py	❌	A 独有
Redis	✅ Session 存储	✅ Session + OTP + Rate Limit	B 用途更广
Alchemy	✅ alchemy_api_key	❌	A 独有（NFT 数据）

结论: B 的 `plib/` 工具库是 A 的**直接复制**，但 A 的 Telegram 集成在 B 中缺失。

7. 代码质量对比

7.1 测试覆盖

项目	Codebase A	Codebase B
单元测试	✅ `test/test_lottery_service.py` (1 个)	❌ 无
集成测试	❌	❌
测试框架	pytest	❌

严重问题: B 没有任何测试文件，规模却是 A 的 4 倍。

7.2 错误处理

Codebase A

- `error.py`: 定义 `UserError`, `ServerError` 枚举
- 大部分错误处理用 `try-except` + `raise UserError.XXX.http()`
- **问题**:
  - `ogger.info`（拼写错误, line 183 in user.py）
  - 部分异常处理过于宽泛（`except Exception`）

Codebase B

- `error.py`: 同样定义 `UserError`, `ServerError`
- 全局异常处理器（`app.py` line 86–96）
  - DEV: 显示详细错误
  - PROD: 只返回“Internal server error”
- **优点**:
  - 更规范的异常处理
  - 全局兜底
  - 日志记录更完善

7.3 已知 Bug

Codebase A

- `user.py` line 183: `ogger.info` 拼写错误（应为 `logger.info`）
- `user.py` line 223: `twitter_id` 未定义（应为 `twitter_result["id"]`）

Codebase B

- 无明显语法错误（已通过基础验证）
- **潜在问题**:
  - 抽卡引擎无测试（概率计算容易出错）
  - Stripe Webhook 签名验证（需审计）
  - 回购价格计算（85% FMV 逻辑需验证）

7.4 代码风格

特性	Codebase A	Codebase B
类型注解	✅ 部分	✅ 完整 (Mapped[])
Docstring	⚠️ 少	✅ 多 (尤其 API)
注释	⚠️ 少	✅ 详细 (包含 PRD 引用)
命名规范	✅ 统一	✅ 统一
代码组织	⚠️ 部分混乱	✅ 清晰分层

结论: B 的代码质量**略优于** A（更规范，但缺测试）。

---

8. 合并建议

8.1 合并可行性分析

结论: 不建议直接合并，建议以 B 为基础，选择性移植 A 的功能。

8.2 推荐方案: 以 B 为基础 (70% 工作量)

原因

1. **架构更完整**: B 实现了 PRD 80%+ 功能（Marketplace, Buyback, Wallet, Admin）
2. **代码更规范**: 全局异常处理 + Rate Limiting + 审计日志
3. **功能更丰富**: 多支付渠道 + Email OTP + Refresh Token
4. **扩展性更好**: 数据模型支持复杂业务（vault, ledger, multi-address）

需要从 A 移植的功能

功能	A 的实现	B 的状态	移植难度
Twitter OAuth	✅ 完整	⚠️ 只有绑定	⭐⭐ (中)
Telegram 集成	✅ ext_service/tg.py	❌ 无	⭐ (低)
树形抽卡引擎	✅ lottery.py	❌ 扁平抽卡	⭐⭐⭐ (高)
单元测试	✅ test_lottery_service.py	❌ 无	⭐⭐ (中)
Alchemy 集成	✅ alchemy_api_key	❌ 无	⭐ (低)

具体步骤

Phase 1: 补全 B 的缺失功能 (2 周)

1. **Twitter OAuth 完整流程** (3 天)
   - 复制 A 的 `oauth.py` 逻辑到 B 的 `auth.py`
   - 更新 `config.py` 加回 Twitter 配置
   - 测试 OAuth 流程
2. **Telegram 集成** (2 天)
   - 复制 A 的 `ext_service/tg.py` 到 B
   - 添加 Telegram Bot Token 配置
   - 实现通知功能（订单状态、回购审批）
3. **Alchemy 集成** (1 天)
   - 加 `alchemy_api_key` 到 config
   - 实现 NFT 元数据查询
4. **测试框架** (5 天)
   - 搭建 pytest 环境
   - 复制 A 的 `test_lottery_service.py` 改写为 `test_pack_engine.py`
   - 新增测试: 认证、支付、市场、回购
5. **Bug 修复** (2 天)
   - 修复 A 的拼写错误 (`ogger`)
   - 审计 B 的抽卡概率计算
   - 审计 Stripe Webhook 签名验证

Phase 2: 数据迁移 (如果 A 有生产数据) (1 周)

1. **用户数据**
   - `users` 表字段映射:
     - A. `sol_address` → B. `wallet_address`
     - A. `twitter_id` + `twitter_name` → B. `twitter_handle`
   - `balances` → B. `User.credit_balance` (积分转货币)
   - `referral_relationships` → B. `User.referred_by`
2. **订单数据**
   - A 的 `orders` (购买盲盒) → B 的 `pack_opening`
   - A 的 `orders` (实物兑换) → B 的 `redemption_order`
   - **需要人工判断**: 根据 `order_type` 分流
3. **NFT 数据**
   - A 的 `nfts` → B 的 `nft` + `user_vault`

- `nft_categories` → 手动创建对应的 `nft` 记录

Phase 3: 前端适配 (2 周)

1. 更新 API 调用
   - `/user/sign-in` → `/auth/wallet/challenge`
   - `/user/verify` → `/auth/wallet/verify`
   - `/product/open` → `/packs/{id}/purchase` + `/packs/openings/{id}/open`
2. 新增页面
   - Marketplace (市场交易)
   - Buyback (回购申请)
   - Wallet (余额充值)
   - Redemption (兑换流程)

Phase 4: 部署上线 (1 周)

1. 环境配置
   - `.env` 文件更新（加 Stripe/Twitter 配置）
   - Redis 配置（增加 OTP/Rate Limit 存储）
2. 数据库迁移
   - 运行 Alembic 迁移脚本（创建新表）
   - 备份 A 的数据库
   - 执行数据迁移脚本
3. 监控
   - 增加 Sentry 错误追踪
   - 配置 Grafana 监控（API 延迟、错误率）

总计: 6 周全职开发时间

8.3 替代方案: 以 A 为基础 (需 100%+ 工作量，不推荐)

如果选择 A，需要重新实现 B 的所有功能: – Marketplace (6 表 + 6 API + 交易引擎) – Buyback (3 表 + 4 API + 审批流程) – Wallet (4 表 + 5 API + Stripe 集成) – Email OTP (Redis + 邮件) – Admin 后台 (10+ API + 审计日志) – 等等...

预估: 10–12 周，且需要从头设计架构。**不推荐**。

8.4 抽卡引擎选择

需求	推荐方案
简单稀有度抽卡	使用 B 的扁平引擎 (pack_drop_table)

需求	推荐方案
复杂多层抽卡 (如: 先抽宝箱类型，再抽内容物)	移植 A 的树形引擎 (unpack_strategies)
保底机制	必须用 B (已实现 pity_counter)
性能要求	B 的扁平引擎更快 (O(n) vs O(n*depth))

建议: – 阶段 1: 用 B 的扁平引擎 (快速上线) – 阶段 2: 可选增强（如需要复杂抽卡，再引入 A 的树形引擎)

---

9. 风险评估

9.1 技术风险

风险	严重性	缓解措施
B 无测试	🔴 高	优先补全测试（尤其抽卡引擎、支付流程)
抽卡概率错误	🔴 高	人工审计 + 单元测试 + 实际抽样验证
Stripe Webhook 漏洞	🔴 高	审计签名验证逻辑 + 添加重放攻击防护
数据迁移失败	🟡 中	多次演练 + 备份 + 灰度迁移
前端 API 不兼容	🟡 中	提前更新前端，发布前充分测试
回购价格计算错误	🟡 中	人工审计 + 单元测试 + 财务审核

9.2 业务风险

风险	严重性	缓解措施
用户数据丢失	🔴 高	迁移前完整备份 + 只读模式验证
订单状态混乱	🟡 中	A/B 订单分流逻辑清晰 + 人工复核
支付流程中断	🟡 中	上线前充分测试 + 回滚方案
NFT 重复铸造	🟡 中	B 的 NFT 预生成机制 + 唯一性约束

9.3 时间风险

里程碑	预估时间	风险
Phase 1 (补全功能)	2 周	🟢 低 (功能清晰)
Phase 2 (数据迁移)	1 周	🟡 中 (取决于 A 的数据量)
Phase 3 (前端适配)	2 周	🟡 中 (取决于前端团队)
Phase 4 (部署上线)	1 周	🟢 低
总计	**6 周**	🟡 中

---

10. 总结与建议

10.1 核心结论

1. **Codebase B (gem-platform) 是更好的基础**
   - 功能覆盖 PRD 80%+
   - 架构更成熟（安全性、扩展性）
   - 代码质量略优（规范性）
2. **两者不可直接合并**
   - 数据模型不兼容（用户表、订单表、NFT 表）
   - API 接口不兼容（路径、参数、流程）
   - 业务逻辑不兼容（抽卡引擎、支付系统、订单流程）
3. **关键缺失 (需补全)**
   - B 缺少 Twitter OAuth 完整流程（只有绑定）
   - B 缺少 Telegram 集成
   - B 缺少测试覆盖
   - B 缺少树形抽卡引擎（如需复杂抽卡）

10.2 行动建议

立即行动 (第 1 周)

1. ✅ 确认 B 为主代码库
2. ✅ 移植 A 的 Twitter OAuth 完整流程
3. ✅ 移植 A 的 Telegram 集成
4. ✅ 搭建测试框架，补全核心测试（抽卡、支付、市场）

中期 (第 2–4 周)

5. ✅ 审计 B 的抽卡引擎概率计算（人工 + 测试）
6. ✅ 审计 Stripe Webhook 签名验证
7. ✅ 审计回购价格计算（85% FMV）
8. ✅ 前端适配新 API（Marketplace, Buyback, Wallet）
9. ✅ 数据迁移脚本开发 + 测试

上线前 (第 5–6 周)

10. ✅ 完整端到端测试（用户注册 → 购买 → 开盒 → 交易 → 回购 → 兑换）
11. ✅ 压力测试（并发购买、抽卡）
12. ✅ 安全审计（SQL 注入、XSS、CSRF、Webhook 签名）
13. ✅ 灰度发布（小流量验证）

10.3 关键决策

需要 CTO/Product 确认:

1. **抽卡引擎选择**
   - 🅰️ 扁平引擎 (B 现有，简单快速)
   - 🅱️ 树形引擎 (A 现有，灵活复杂)
   - 🅾️ 混合 (Phase 1 用扁平，Phase 2 增强为树形)
2. **数据迁移策略**
   - 🅰️ 硬切换 (停服 → 迁移 → 上线)
   - 🅱️ 双写 (同时写 A/B，逐步迁移流量)
   - 🅾️ 新老系统并存 (A 只读，新订单走 B)
3. **上线时间**
   - 🅰️ 尽快上线 (4 周，功能不完整)
   - 🅱️ 稳妥上线 (6 周，推荐)
   - 🅾️ 完美上线 (8 周，包含压力测试 + 安全审计)

10.4 最终建议

推荐方案: 以 B 为基础 + 6 周开发周期 + 稳妥上线

理由: – B 的架构已覆盖主要业务（MVP++） – 6 周足够补全缺失功能 + 测试 – 风险可控（有备份 + 灰度发布）

不推荐: 以 A 为基础（需重复造轮子，时间翻倍）

附录: 文件清单

Codebase A (gema-backend-main)

```
app/
├── api/
│   ├── user.py (258 lines)
│   └── product.py (~150 lines, 未读)
├── models/
│   ├── user.py (66 lines) - User, Balance, RewardRecord, ReferralRelationship
│   ├── order.py (107 lines) - Order, Nft, ShippingAddress, ShippingTrack,
Payment
│   ├── product.py (97 lines) - Product, NftCategory, UnpackProbability,
UnpackStrategy, ShippingFee
│   └── base.py (10 lines) - BaseTable
├── services/
│   ├── user.py
│   ├── order.py
│   ├── product.py
│   ├── token.py
│   ├── sol_api.py
│   ├── ton_api.py
│   ├── tg.py
│   └── error.py
├── plib/
│   ├── web3_sol.py - Solana 签名验证
│   ├── web3_ton.py - TON 签名验证
│   ├── oauth.py - Twitter OAuth
│   ├── sendmail.py - SendGrid
│   ├── address_api.py - Shippo
│   ├── session_db.py - Redis Session
│   ├── session_store.py
│   ├── utils.py
│   └── local_api.py
├── ext_service/
│   └── tg.py - Telegram Bot
├── test/
│   └── test_lottery_service.py
├── app.py (49 lines)
├── config.py (41 lines)
└── enums.py (63 lines)
```

Codebase B (gem-platform/backend)

```
app/
├── api/
│   ├── auth.py (232 lines) — Wallet + Email OTP + Twitter Bind
│   ├── user.py (66 lines) — GET/PATCH /users/me
│   ├── pack.py (~200 lines) — 盲盒 CRUD + 购买 + 开盒
│   ├── vault.py (86 lines) — 用户仓库查询
│   ├── market.py (~150 lines) — Marketplace 交易
│   ├── buyback.py (~100 lines) — 回购申请
│   ├── wallet.py (~150 lines) — 钱包充值 + 账本
│   ├── order.py (~200 lines) — 兑换订单
│   ├── referral.py (~80 lines) — 推荐系统
│   ├── rank.py (~60 lines) — 排行榜
│   ├── admin.py (~300 lines) — 管理后台
│   ├── webhooks.py (~100 lines) — Stripe Webhook
│   └── nft.py (~150 lines) — NFT CRUD (legacy)
├── models/
│   ├── user.py (36 lines) — User
│   ├── pack.py (170 lines) — Pack, PackDropTable, PackOpening, UserVault
│   ├── pack_version.py
│   ├── nft.py (76 lines) — Nft, NftCollectionMeta, NftSequence, NftPayment
│   ├── order.py (142 lines) — RedemptionOrder, RedemptionOrderItem,
ShippingAddress
│   ├── market.py (39 lines) — MarketListing, MarketOffer
│   ├── buyback.py (19 lines) — BuybackRequest
│   ├── wallet.py (105 lines) — Wallet, Deposit, WalletLedger
│   ├── referral.py (35 lines) — ReferralReward
│   ├── admin_audit.py (28 lines) — AdminAuditLog
│   ├── community.py (15 lines) — Community
│   ├── enums.py (183 lines) — 各种枚举
│   └── base.py (10 lines)
├── services/
│   ├── auth.py — 认证服务
│   ├── pack.py — 盲盒服务
│   ├── pack_engine.py — 抽卡引擎
│   ├── market.py — 市场服务
│   ├── buyback.py — 回购服务
│   ├── wallet_payment.py — 钱包支付
│   ├── payment.py — 支付通用
│   ├── order.py — 订单服务
│   ├── referral.py — 推荐服务
│   ├── rank.py — 排行榜
│   ├── security.py — 安全检查
│   ├── stripe_webhook.py — Stripe 事件处理
│   ├── admin_*.py — 管理后台服务
│   └── error.py
├── plib/ (与 A 相同，复制过来)
├── app.py (121 lines)
├── config.py (45 lines)
└── (无测试)
```

报告生成时间: 2026-02-11 **分析用时**: 约 20 分钟（读取 108 个文件） **建议审阅者**: CTO + Product Manager + Lead Backend Engineer