Техническое задание (ТЗ)

Проект: Многосторонний маркетплейс (аналог Wildberries) с картой рынков/магазинов, чатом покупатель–продавец, каталогом, заказами, оплатой, доставкой, рекламой, аналитикой и подписками продавцов.
Локация/валюта: Бишкек (Кыргызстан), KGS.
Языки: ru (основной), kg, en, zh, kz, uzb (поля предусмотрены, перевод постепенно).
Стек: Backend — Java 21, Spring Boot 3.x, PostgreSQL 15+, Redis, Kafka; Frontend — Angular 17+; Mobile — Flutter 3.x (Android/iOS); Поиск — OpenSearch/Elasticsearch; Хранилище — S3-совместимое (MinIO).
Архитектура: модульный монолит с возможностью выделения доменов в сервисы (Search, Payment, Ads, Delivery).

---

1. Цели и показатели

- MVP за 3 месяца: ≥100 продавцов, ≥500 SKU, карта рынков.
- Поиск <800мс, uptime 99.5%.
- Покупка возможна только у магазинов с активной **Смарт-POS** и включенным payout-аккаунтом.
- В будущем — доставка, фулфилмент, интеграция с СНГ, реклама, аналитика уровня Wildberries.

---

2. Роли

- **Покупатель:** регистрация, поиск, просмотр, чат, покупка, отзывы.
- **Продавец:** добавление товаров, управление остатками, аналитика, реклама, POS.
- **Администратор:** модерация, отчеты, управление тарифами, комиссиями, поддержка.
- **Платформа:** аналитика, антифрод, расчёты, модерация, реклама.

---

3. Ключевые особенности и бизнес-правила

3.1 Смарт-POS

- Устройство продавца для сканирования товара при продаже (торговая смарт-касса с POS-сканером).
- При каждой продаже POS синхронизирует данные с БД: количество, цена, скидки, возвраты.
- Система автоматически обновляет наличие товара без участия продавца.
- Если POS неактивен >10 минут — товары временно теряют возможность онлайн-покупки.

3.2 Социальная витрина (Instagram-подобный UX)

- **Две ключевые ленты:**
    1. **Подписки (Following):** публикации продавцов и рынков, на которых подписан пользователь.
    2. **Поиск/Explore:** бесконечная лента с миксом публикаций (видео/картинки), ранжирование по релевантности/популярности.
- **Карта:** отдельный режим/оверлей, где публикации и магазины привязаны к гео-точкам, с быстрым переходом к карточке товара/места.
- **UGC типы:** пост (1..10 фото/видео), карусель, короткие видео (≤60 сек), теги/хэштеги, закрепы.
- **Действия:** лайк, сохранить, поделиться, подписаться/отписаться, жалоба, скрыть.
- **Реклама:** промо-посты/бусты в лентах с пометкой «Реклама».
- **SEO/Доступность:** alt-тексты к медиа, субтитры для видео.

3.3 Комментарии и отзывы (политики)

- **Опции платформы (конфигурируемо):** `comments_mode = all | buyers_only | followers_only | off`.

    - `buyers_only` — комментировать могут только пользователи с **верифицированной покупкой** (значок рядом с именем).
    - `all` — любой зарегистрированный пользователь (по умолчанию, если доля магазинов без POS высока).
    - `followers_only` — только подписчики владельца поста.
    - `off` — комментарии отключены.

- **Верификация покупки в комментариях:** `verified_purchase=true`, если есть заказ по продавцу/товару.
- **Модерация:** автофлаги, стоп-слова, жалобы, теневое ограничение (shadow-ban), блок списки.

3.4 Buy Eligibility (право на онлайн-покупку)

- `online_purchase`: Смарт-POS активен и payout включен — показывается кнопка **«Купить»**, активна корзина и оплата.
- `manual_contact`: POS неактивен или нет payout — кнопка **«Купить»** скрыта; доступны **«Связаться», «Позвонить», «Запросить наличие»** (авто-сообщение).

3.5 Финансы и комиссии

- Комиссия площадки (take rate): хранится в БД, по умолчанию 0 (платформа бесплатна).
- Плательщик эквайринга: продавец.
- Выплаты продавцам: **моментальные** (после успешной онлайн-покупки).
- Возвраты: только полные (без частичных), ответственность (платформа/продавец) — по доказательствам.

3.6 Подписки продавцов

- Тарифы: **FREE / PRO / ENTERPRISE**; лимиты SKU, бусты в поиске, доступ к Ads.
- Триал 14 дней, автопродление, можно отключить.

3.7 География и локализация

- Этап 1: Бишкек → Этап 2: Ош/регионы КР → Этап 3: СНГ.
- Языки: ru (первый релиз), далее kg, en, zh, kz, uzb.

3.8 Самовывоз и доставка

- Самовывоз со слотами; доставка — партнёрская, подтверждение кодом.
- Фулфилмент-центр — не в MVP.

3.9 POS-данные, цены и инвентарь

- События POS: продажи, возвраты, аннулирования, скидки, наличные.
- Онлайн и кассовая цена идентичны.
- Обязательны штрих-коды (EAN/UPC/QR).
- Поддержка наборов (bundles).
- Резервирование: 20–30 мин до оплаты, 24ч после оплаты до самовывоза.

3.10 Реклама и аналитика

- Модели: CPC/CPM/фикс; frequency cap 3–5/день/пользователь.
- Метрики магазинов и точек на карте; экспорт CSV/XLSX.
- Антифрод кликов и спама в чате/комментариях.

---

4. Архитектура и компоненты

Модули: Auth, Merchant, Catalog, Inventory, Order, Payment, Billing, Delivery, Ads, Analytics, POS, Moderation, **Social (UGC/Feed/Follow)**.
Коммуникации: REST + WebSocket, Kafka (события), Redis pub/sub (реактив).

4.1 Взаимодействие модулей (обзор)

- Пользователь → Front (Angular/Flutter) → API (Spring Boot)

- → Auth → Merchant → Catalog → Inventory → POS → Order → Payment → Billing → Analytics
- ↘ Social(Feed/UGC/Follow) ↙ ↘ Delivery ↙ ↘ Ads ↙

- **Social** хранит публикации, лайки, подписки, комментарии; даёт ленты Following/Explore и интеграцию с картой.
- **POS** обновляет Inventory → влияет на кнопку «Купить» и на Explore (активные товары).
- **Order/Payment/Billing** обеспечивают моментальные выплаты.

4.2 Данные (PostgreSQL) — дополнение для Social

- `post(id, owner_type ENUM[user,seller], owner_id, text, created_at, geo{lat,lon}, visibility ENUM[public,followers,private], status)`
- `post_media(id, post_id, type ENUM[image,video], url, thumb_url, duration_s, width, height, sort)`
- `follow(follower_user_id, target_type ENUM[seller,user], target_id, created_at)`
- `like(user_id, post_id, created_at)`
- `comment(id, post_id, user_id, text, created_at, is_flagged, verified_purchase bool)`
- `post_tag(post_id, tag)` / `post_place(post_id, place_id)`
- Индексы: GIN по тексту и тегам, гео-индексы по `geo` для карты.

4.3 Медиа-пайплайн

- Загрузка в S3/MinIO, очереди на обработку (Kafka).
- Транскодирование видео (FFmpeg workers): MP4/HLS, превью, постеры, генерация разных разрешений.

- CDN-кеширование, лимиты размера (фото ≤10 МБ, видео ≤200 МБ), антивирус/безопасность.

4.4 API ключевые

Social/Feed

- `GET /feed/following?page=` — лента подписок.
- `GET /feed/explore?page=&tags=&near=lat,lon` — лента Explore (рекомендации + по карте).
- `POST /posts` / `GET /posts/{id}` / `DELETE /posts/{id}`.
- `POST /posts/{id}/like` / `DELETE /posts/{id}/like`.
- `GET /posts/{id}/comments` / `POST /posts/{id}/comments`.
- `POST /follow` / `DELETE /follow`.
- `GET /map/posts?bbox=` — публикации на карте.
- **Политика комментариев:** сервер учитывает `comments_mode` платформы/продавца и `verified_purchase`.

Core (без изменений ключевых):

- `/merchants/{id}/payout/onboard`, `/merchants/{id}/eligibility`, `/pos/sales`, `/orders`, `/analytics/*`.

4.5 Диаграммы (Mermaid)

Лента Following/Explore

- sequenceDiagram
- participant U as User
- participant FE as Frontend
- participant API as Social API
- participant R as Recommender
- U->>FE: Открывает Explore
- FE->>API: GET /feed/explore?near=lat,lon
- API->>R: запрос кандидатов (сигналы: интересы, популярность, гео)
- R-->>API: список постов
- API-->>FE: посты (медиа, владельцы, метрики)

Комментарии (buyers_only)

- sequenceDiagram
- FE->>API: POST /posts/{id}/comments
- API-->>API: Проверка comments_mode + verified_purchase
- API-->>FE: 403 если нет права; 200 если ок

4.6 Ранжирование Explore (минимум)

- Сигналы: вовлечённость (лайки/комменты/сохранения), свежесть, гео-близость, релевантность тегам/категориям, качество продавца, CTR.

- Антиспам: лимиты публикаций, повторов, NSFW-фильтры.

- `/merchants/{id}/payout/onboard` — онбординг выплат.

- `/merchants/{id}/eligibility` — флаг возможности покупки.

- `/pos/sales` — приём POS-продаж (идемпотентный).

- `/orders` — проверка eligibility, при несоответствии → `409 BUY_NOT_ELIGIBLE`.

- `/analytics/shops`, `/analytics/products` — отчёты.

4.4 Диаграммы (Mermaid + ASCII)

Context (C4 level 1)

- flowchart TB
- User[Покупатель] --> Web[Angular Web]
- User --> Mobile[Flutter App]
- Seller[Продавец] --> SellerApp[Flutter Seller App]
- Web --> API[Spring Boot API]
- Mobile --> API
- SellerApp --> API
- API --> PG[(PostgreSQL)]
- API --> Cache[(Redis)]
- API --> S3[(MinIO/S3)]
- API --> Search[(OpenSearch)]
- API <--> MQ[(Kafka)]

- MQ --> Analytics[(Analytics/ClickHouse)]
- API --> PSP[(PSP/Payouts)]
- API --> Map[Maps SDK]

POS → Инвентарь → Витрина (последовательность)

- sequenceDiagram
- participant POS as Смарт-POS
- participant API as API /pos
- participant INV as Inventory
- participant IDX as Search Indexer
- participant FE as Frontend
- POS->>API: POST /pos/sales {items, external_ref}
- API->>INV: create inventory_adjustment(-qty)
- INV-->>IDX: event inventory.updated
- IDX-->>FE: обновлённая карточка товара (остатки)

Заказ → Оплата → Выплата (последовательность)

- sequenceDiagram
- participant U as Покупатель
- participant FE as Web/Mobile
- participant API as Orders API
- participant PSP as PSP
- participant BILL as Billing
- participant SELLER as Продавец
- U->>FE: Купить
- FE->>API: POST /orders
- API-->>API: Проверка buy_eligibility & payout_status
- API->>PSP: Создать платёж
- PSP-->>API: webhook payment.succeeded
- API->>BILL: Выплата продавцу (instant payout)
- BILL->>SELLER: Уведомление о поступлении

Состояния Buy Eligibility

- stateDiagram-v2
- [*] --> manual_contact
- manual_contact --> online_purchase: posEnabled & payout=active
- online_purchase --> manual_contact: POS heartbeat > 10m

ASCII Data Flow (fallback)

- Seller POS --> /pos/sales --> Inventory.adjust --> Kafka.event --> Search.index --> Frontend
- Buyer --> Web/Mobile --> Orders --> PSP --> Billing (instant payout) --> Seller

---

5. Аналитика

- События: view, click, add_to_cart, route_built, chat_started, order_created, pos_sale_recorded, **post_view, post_like, post_comment, follow, share, save**.
- Отчёты: по магазину, по постам (ER, reach, CTR), по карте (просмотры точки, построение маршрута).
- Экспорт: CSV/XLSX.
- UTM/рефералы, анонимизация user_id, opt-out.

6. Тестирование и контроль качества Тестирование и контроль качества

- Юнит/интеграционные (JUnit/Testcontainers).
- e2e: Angular + Playwright, Flutter integration tests.
- Нагрузочные: Gatling/k6, 200RPS каталог, 50RPS поиск.
- Проверка Buy Eligibility (UI и API).
- Проверка POS heartbeat >10 мин → скрытие «Купить».
- Проверка выплат (моментальные переводы).
- Проверка аналитики (агрегации, дедупликация).
- Антифрод и антиспам тесты.

---

7. Развитие и масштабирование

- Подключение налоговых касс.
- Централизованный склад (фулфилмент).
- Международная экспансия (СНГ).
- Автоматизация доставки.
- Расширение рекламы и аналитики.

Итог: ТЗ включает все бизнес-правила, POS-интеграцию, моментальные выплаты, Buy Eligibility, аналитику, подписки, рекламу, локализацию, безопасность и масштабируемость. Документ готов к передаче разработчикам и дизайнерам для детальной проработки.

13. Технические и эксплуатационные параметры

- Размещение: локальный дата-центр в Бишкеке.
- **Деплой:** Linux + Docker. На старте — вручную (`git pull` + `docker compose up -d`); БД поднимается на отдельном хосте (вне compose).
- **CI/CD:** GitLab CI/CD (позже — автоматизация rollout), приватный реестр образов.
- Массовая загрузка товаров CSV/Excel/XML.
- Интеграция с 1C/ERP через API.
- Индексация в поиске ≤5 секунд.
- Антиспам/лимиты в чате и комментариях.
- POS-API rate limit и heartbeat-мониторинг.
- **Data Retention:** сырые события — 13 мес.; медиа — без срока; логи — 90 дней.

14. Архитектура и безопасность (новые уточнения)

14.1 Архитектура backend

- Слойность: `Controller → Service → Repository`.
- Используется Spring Boot со строгим разделением по доменам (`auth`, `catalog`, `inventory`, `social`, и т.д.).
- DTO/Entity маппинг — **MapStruct**; валидация — **Jakarta Validation**.
- API документируется через **OpenAPI 3 / Swagger UI**.
- Exception handling — глобальный через `@ControllerAdvice`.

14.2 Безопасность

- Двухфакторная аутентификация (2FA) обязательна для **продавцов и администраторов** (email/TOTP).
- Пароли хранятся с хешированием **BCrypt**.
- Аутентификация через **JWT** с refresh-токенами.
- **Audit Log:** все действия в админке и кабинете продавца логируются (создание, изменение, удаление, публикации, модерация, смена статусов).
- События аудита хранятся в отдельной таблице `audit_log` (actor_id, role, action, entity, timestamp, ip).

15. Карта и геоданные

- Источники карт: **Google Maps API** и **2GIS API**.
- Приоритет — точность и быстрота отклика; локальное кэширование тайлов/маршрутов может быть добавлено позже.
- Магазины и рынки имеют гео-точки с координатами; реализована кластеризация точек и построение маршрута.
- Публикации и магазины связаны с гео-координатами; возможен фильтр по расстоянию.

16. Категории и модерация контента

- Централизованная структура категорий, аналогичная Wildberries.

- Продавцы выбирают категории из утверждённого справочника.
- Новые категории добавляются только администраторами.
- Контент (посты, медиа, описания) проходит автоматическую и ручную модерацию.

17. Реклама и продвижение (Boost)

- В будущем будет поддержка **Boost-постов**: продавцы могут продвигать свои публикации в ленте Explore.
- Для продвижения будет создан **внутренний баланс/кошелёк** в модуле Billing.
- Сценарии: пополнение баланса, списание за показы/клики, отчёты по эффективности кампаний.

18. Аналитика и фильтры

- События аналитики фиксируются при действиях клиента с магазином (просмотр магазина, просмотр товара, открытие маршрута, переход по карте).
- Отчёты продавца включают фильтры по источнику перехода (карта, лента, поиск).
- Метрики: посещаемость, клики по контактам, построенные маршруты, взаимодействия с товарами.

19. Мониторинг и DevOps

- В дальнейшем подключаются **Prometheus + Grafana** для метрик (CPU, RPS, задержки Kafka, POS heartbeat, ошибки API).
- Настраивается **Alerting** на сбои POS, задержки heartbeat, ошибки выплат, превышение SLA по API.
- Мониторинг подключается на production-этапе.

20. Мобильное приложение (Flutter)

- Один Flutter-клиент с возможностью переключения между ролями **«Покупатель»** и **«Продавец»**.

- Для продавца доступны отдельные разделы: *Товары*, *Продажи*, *POS-состояние*, *Аналитика*.

- Реализована поддержка **push-уведомлений** (новые заказы, чаты, комментарии, сбои POS, обновления аналитики).

- Для покупателей — стандартные уведомления о заказах, ответах продавцов и обновлениях публикаций.

21. ERD – Архитектура данных

21.1 Общая схема (Mermaid)

- erDiagram
-
- USER {
- uuid id PK
- varchar phone
- varchar email
- varchar password_hash
- varchar role "buyer | seller | admin"
- bool is_verified
- timestamp created_at
- }
-
- MERCHANT {
- uuid id PK
- uuid owner_id FK "→ USER.id"
- varchar name
- varchar description
- varchar payout_account_number nullable
- varchar payout_bank_name nullable
- varchar payout_status "pending | active | suspended"
- varchar buy_eligibility "manual | pos | hybrid"
- jsonb settings_json
- timestamp created_at
- }
-
- SHOP {
- uuid id PK
- uuid merchant_id FK "→ MERCHANT.id"
- varchar name
- varchar address

```
    varchar phone
    numeric lat
    numeric lon
    varchar pos_status "inactive | active"
    timestamp created_at
  }

  CATEGORY {
    uuid id PK
    varchar name
    uuid parent_id nullable
    int sort_order
  }

  PRODUCT {
    uuid id PK
    uuid shop_id FK "→ SHOP.id"
    uuid category_id FK "→ CATEGORY.id"
    varchar name
    text description
    bool is_active
    timestamp created_at
  }

  PRODUCT_VARIANT {
    uuid id PK
    uuid product_id FK "→ PRODUCT.id"
    varchar sku
    varchar barcode
    jsonb attributes_json "size/color/etc"
    numeric price
    int stock_qty
    timestamp updated_at
  }

  POS_SALE {
    uuid id PK
    uuid shop_id FK "→ SHOP.id"
    uuid variant_id FK "→ PRODUCT_VARIANT.id"
    numeric qty
    numeric unit_price
    numeric total_price
    varchar receipt_number
    varchar type "sale | refund"
    timestamp sold_at
  }

  POS_SUMMARY_DAILY {
```

```
    uuid id PK
    uuid shop_id FK
    date day
    numeric total_sales
    numeric total_refunds
    int total_receipts
  }

  ORDER {
    uuid id PK
    uuid user_id FK "→ USER.id"
    uuid shop_id FK "→ SHOP.id"
    varchar status "pending | paid | cancelled | fulfilled"
    numeric total_amount
    timestamp created_at
    timestamp paid_at
  }

  ORDER_ITEM {
    uuid id PK
    uuid order_id FK "→ ORDER.id"
    uuid variant_id FK "→ PRODUCT_VARIANT.id"
    int qty
    numeric price
    numeric subtotal
  }

  POST {
    uuid id PK
    uuid owner_id FK "→ MERCHANT.id or USER.id"
    varchar owner_type "merchant | user"
    text text
    varchar type "photo | video"
    jsonb geo "lat/lon"
    timestamp created_at
    bool is_active
  }

  POST_MEDIA {
    uuid id PK
    uuid post_id FK "→ POST.id"
    varchar type "image | video"
    varchar url
    varchar thumb_url
    int sort_order
    numeric duration_s nullable
  }

```

- COMMENT {
- uuid id PK
- uuid post_id FK "→ POST.id"
- uuid user_id FK "→ USER.id"
- text text
- bool verified_purchase
- bool is_flagged
- timestamp created_at
- }
-
- FOLLOW {
- uuid id PK
- uuid follower_id FK "→ USER.id"
- uuid target_id FK "→ MERCHANT.id or USER.id"
- varchar target_type "merchant | user"
- timestamp created_at
- }
-
- AUDIT_LOG {
- uuid id PK
- uuid actor_id
- varchar actor_role
- varchar action
- varchar entity
- varchar entity_id
- timestamp created_at
- varchar ip
- }
-
- AD_CAMPAIGN {
- uuid id PK
- uuid merchant_id FK "→ MERCHANT.id"
- varchar name
- varchar type "boost | banner | search"
- numeric budget
- varchar status "active | paused | finished"
- timestamp created_at
- }
-
- SHOP_METRIC_DAILY {
- uuid id PK
- uuid shop_id FK "→ SHOP.id"
- date day
- int views
- int clicks
- int route_builds
- int chats_started
- int follows

```
        numeric revenue
    }

    PRODUCT_METRIC_DAILY {
        uuid id PK
        uuid variant_id FK
        date day
        int views
        int clicks
        int add_to_cart
        int orders
        numeric revenue
    }

    %% RELATIONSHIPS
    USER ||--o{ MERCHANT : owns
    MERCHANT ||--o{ SHOP : has
    SHOP ||--o{ PRODUCT : sells
    PRODUCT ||--o{ PRODUCT_VARIANT : has
    SHOP ||--o{ POS_SALE : records
    SHOP ||--o{ POS_SUMMARY_DAILY : aggregates
    USER ||--o{ ORDER : places
    ORDER ||--|{ ORDER_ITEM : contains
    PRODUCT_VARIANT ||--o{ ORDER_ITEM : referenced
    MERCHANT ||--o{ POST : publishes
    POST ||--o{ POST_MEDIA : contains
    POST ||--o{ COMMENT : has
    USER ||--o{ COMMENT : writes
    USER ||--o{ FOLLOW : follows
    MERCHANT ||--o{ AD_CAMPAIGN : runs
    SHOP ||--o{ SHOP_METRIC_DAILY : tracked
    PRODUCT_VARIANT ||--o{ PRODUCT_METRIC_DAILY : tracked
```

21.2 ASCII структура (основные связи)

```
USER (id PK)
 ├─< MERCHANT.owner_id
 ├─< ORDER.user_id
 ├─< COMMENT.user_id
 └─< FOLLOW.follower_id

MERCHANT (id PK)
 ├─< SHOP.merchant_id
 ├─< POST.owner_id (owner_type=merchant)
 ├─< AD_CAMPAIGN.merchant_id
 └─ payout_account_number / bank_name nullable
```

-
- SHOP (id PK)
- ├─< PRODUCT.shop_id
- ├─< POS_SALE.shop_id
- ├─< POS_SUMMARY_DAILY.shop_id
- ├─< ORDER.shop_id
- └─< SHOP_METRIC_DAILY.shop_id
-
- PRODUCT (id PK)
- ├─< PRODUCT_VARIANT.product_id
- └─ category_id → CATEGORY.id
-
- PRODUCT_VARIANT (id PK)
- ├─< ORDER_ITEM.variant_id
- ├─< POS_SALE.variant_id
- └─< PRODUCT_METRIC_DAILY.variant_id
-
- ORDER (id PK)
- └─< ORDER_ITEM.order_id
-
- POST (id PK)
- ├─< POST_MEDIA.post_id
- └─< COMMENT.post_id
-
- COMMENT (id PK)
- FOLLOW (id PK)
- AUDIT_LOG (id PK)
- AD_CAMPAIGN (id PK)
- SHOP_METRIC_DAILY (id PK)
- PRODUCT_METRIC_DAILY (id PK)

21.3 Рекомендации по БД

- Индексы: `btree(shop_id)` на большинстве таблиц, `gin(attributes_json)` для поиска по атрибутам.

- Геоиндекс по координатам (PostGIS).

- Каскадное удаление (`ON DELETE CASCADE`) для зависимых таблиц (`product_variant`, `order_item`, `post_media`).

- Партиционирование по дате в таблицах `pos_sale` и `*_metric_daily`.

- Возможен вынос `audit_log` в отдельную БД при росте объёмов.

22. ERD – Аналитика и метрики

22.1 Общая схема аналитических таблиц

- erDiagram
-
- POS_SALE {
- uuid id PK
- uuid shop_id FK
- uuid variant_id FK
- numeric qty
- numeric total_price
- varchar type "sale | refund"
- timestamp sold_at
- }
-
- ORDER {
- uuid id PK
- uuid shop_id FK
- uuid user_id FK
- numeric total_amount
- varchar status
- timestamp created_at
- timestamp paid_at
- }
-
- ORDER_ITEM {
- uuid id PK
- uuid order_id FK
- uuid variant_id FK
- int qty
- numeric price
- }
-
- AD_CAMPAIGN {
- uuid id PK
- uuid merchant_id FK
- varchar type "boost | banner | search"
- numeric budget
- varchar status
- timestamp created_at
- }
-
- SHOP_METRIC_DAILY {
- uuid id PK
- uuid shop_id FK
- date day

```
        int views
        int clicks
        int route_builds
        int chats_started
        int follows
        numeric revenue
    }

    PRODUCT_METRIC_DAILY {
        uuid id PK
        uuid variant_id FK
        date day
        int views
        int clicks
        int add_to_cart
        int orders
        numeric revenue
    }

    AD_EVENT_DAILY {
        uuid id PK
        uuid campaign_id FK "→ AD_CAMPAIGN.id"
        date day
        int impressions
        int clicks
        numeric spend
        numeric cpc
        numeric cpm
    }

    ANALYTIC_EVENT_RAW {
        uuid id PK
        varchar event_type "view | click | like | follow | order_created |
pos_sale_recorded"
        uuid user_id
        uuid target_id
        varchar target_type "shop | product | post | ad"
        jsonb context
        timestamp created_at
    }

    %% RELATIONS
    SHOP ||--o{ SHOP_METRIC_DAILY : aggregates
    PRODUCT_VARIANT ||--o{ PRODUCT_METRIC_DAILY : aggregates
    AD_CAMPAIGN ||--o{ AD_EVENT_DAILY : aggregates
    POS_SALE ||--o{ SHOP_METRIC_DAILY : contributes
    ORDER ||--o{ SHOP_METRIC_DAILY : contributes
    ORDER_ITEM ||--o{ PRODUCT_METRIC_DAILY : contributes
```

- ANALYTIC_EVENT_RAW ||--o{ SHOP_METRIC_DAILY : feeds
- ANALYTIC_EVENT_RAW ||--o{ PRODUCT_METRIC_DAILY : feeds

22.2 Описание потоков данных

- **POS → Metrics:** каждое событие продажи из POS увеличивает `revenue`, `orders`, `qty_sold`.
- **ORDER → Metrics:** при оплате заказа обновляются `orders`, `revenue` для магазина и товара.
- **AD_CAMPAIGN → AD_EVENT_DAILY:** данные собираются из трекинга показов/кликов рекламы.
- **ANALYTIC_EVENT_RAW:** единый лог всех событий (просмотры, клики, подписки, лайки, заказы).
- Агрегации по расписанию (каждые 5 мин и ежедневно) формируют `*_metric_daily`.

22.3 ASCII схема аналитики

- POS_SALE --→ SHOP_METRIC_DAILY (revenue, orders)
- ORDER --→ SHOP_METRIC_DAILY (orders, revenue)
- ORDER_ITEM --→ PRODUCT_METRIC_DAILY (orders, revenue, qty)
- AD_CAMPAIGN --→ AD_EVENT_DAILY (impressions, clicks, spend)
- ANALYTIC_EVENT_RAW --→ SHOP_METRIC_DAILY / PRODUCT_METRIC_DAILY (views, clicks, follows)

22.4 Рекомендации по аналитике

- Сырые события хранятся в `analytic_event_raw` (13 мес).
- Ежедневные агрегации — `*_metric_daily` (хранятся бессрочно).
- Методы агрегации: `materialized view` + `cron` обновления.
- Индексы: `btree(day, shop_id)`, `btree(day, variant_id)`, `gin(context)`.
- Возможность экспорта в CSV/XLSX через API `/analytics/export`.

23. Data Pipeline – поток аналитики и событий

23.1 Источники событий

- **Frontend (Angular / Flutter):** генерирует события `view`, `click`, `follow`, `like`, `comment`, `add_to_cart`, `route_built`, `chat_started`, `order_created`.
- **Backend (Spring Boot):** создаёт системные события `order_paid`, `refund_created`, `payout_made`, `pos_sale_recorded`.
- **POS Devices:** отправляют транзакционные события `pos_sale` и `pos_refund` напрямую в API.

23.2 Поток обработки

- flowchart LR
- FE[Frontend apps] --> API[Spring Boot API]
- POS[Smart-POS devices] --> API
- API --> KAFKA[(Kafka topics)]
- KAFKA --> ETL[ETL Worker / Kafka Consumer]
- ETL --> PG_RAW[(PostgreSQL analytic_event_raw)]
- ETL --> AGG[Aggregator Jobs (cron)]
- AGG --> METRICS[(shop_metric_daily, product_metric_daily, ad_event_daily)]
- METRICS --> API_EXPORT[/Analytics API & Reports/]

23.3 Этапы обработки

Этап	Компонент	Описание
Ingestion	Kafka	Потоковая приёмка событий из API и POS. Используются топики: `analytics.raw`, `pos.sales`, `ads.events`.
Storage	PostgreSQL	Таблица `analytic_event_raw` хранит сырые события (13 мес).

Aggregation	Cron / Spring Scheduler	Каждые 5 минут агрегирует данные в `*_metric_daily`.
Metrics Update	Shop/Product/Ad	Обновляет ключевые показатели: просмотры, клики, заказы, доход.
Reporting	Analytics API	Доступ через `/analytics/shops`, `/analytics/products`, `/analytics/export`.

23.4 Пример последовательности событий

- sequenceDiagram
- participant U as User
- participant FE as Flutter/Angular
- participant API as API (Spring Boot)
- participant K as Kafka
- participant DB as PostgreSQL
- participant CRON as Aggregator
-
- U->>FE: просмотр товара
- FE->>API: POST /events { type:view, target:product }
- API->>K: publish analytics.raw
- K-->>API: ack
- API->>DB: log minimal event
- CRON->>DB: aggregate hourly → update product_metric_daily
- DB-->>Reports: данные доступны в отчётах

23.5 Формат событий

Поле	Тип	Пример	Комментарий
`event_id`	UUID	2c7f...	Уникальный идентификатор

`event_type`	ENUM	view/click/order_created/...	Тип события
`user_id`	UUID	...	Источник (если авторизован)
`target_id`	UUID	...	ID магазина, товара, поста
`target_type`	ENUM	shop/product/post/ad	Цель события
`context`	JSONB	{"page":"explore","device":"ios"}	Дополнительная информация
`created_at`	TIMESTAMP	...	Время генерации события

23.6 Рекомендации

- Использовать **batch insert** для ETL и **upsert** при агрегации.
- Добавить **TTL / архивирование** для `analytic_event_raw` старше 13 месяцев.
- При росте нагрузки — вынести агрегации в отдельный микросервис `analytics-worker`.
- Возможен переход на **ClickHouse** для OLAP-аналитики (опционально).

24. POS и Инвентаризация — поток данных и синхронизация

24.1 Компоненты POS-интеграции

- **Смарт-POS устройство:** отправляет продажи, возвраты, остатки и регулярные heartbeats.
- **POS API (Spring Boot):** принимает данные, проверяет подпись и токен устройства, обновляет инвентарь.
- **Inventory Service:** пересчитывает остатки и публикует события `inventory.updated` в Kafka.
- **Search Indexer:** слушает события и обновляет карточки товаров на витрине и карте.

24.2 Последовательность обмена

- sequenceDiagram
- participant POS as Смарт-POS
- participant API as POS API (Spring Boot)
- participant INV as Inventory Service
- participant IDX as Search Indexer
- participant FE as Frontend
-
- POS->>API: POST /pos/heartbeat { device_id, status, stock[] }
- API-->>POS: 200 OK (ack)
- POS->>API: POST /pos/sales { items, receipt_number }
- API->>INV: update stock (-qty)
- INV-->>IDX: publish inventory.updated
- IDX-->>FE: push обновление карточки товара
- Note over FE: UI обновляет остаток и доступность “Купить”

24.3 Heartbeat-механизм

Интервал	Событие	Действие
каждые 30 сек	`heartbeat`	POS сообщает о доступности и базовых остатках

> 10 мин без heartbeat	API ставит магазин в состояние `pos_offline`
при восстановлении	статус `pos_active` и возобновление синхронизации

24.4 Логика синхронизации остатков

1. При `pos_sale` количество товара (`stock_qty`) уменьшается.
2. При `pos_refund` — увеличивается.
3. API обновляет `product_variant.stock_qty` и публикует событие `inventory.updated`.
4. Search Index получает обновление и обновляет данные в поиске и на карте.
5. Клиенты видят изменение в течение ≤ 5 секунд.

24.5 Форматы данных POS

POS Heartbeat

```
{
  "device_id": "POS-1234",
  "status": "active",
  "timestamp": "2025-11-03T12:00:00Z",
  "stock": [
    {"sku": "A100", "qty": 12},
    {"sku": "A101", "qty": 8}
  ]
}
```

POS Sale

```
{
  "device_id": "POS-1234",
  "receipt_number": "R-98212",
  "items": [
    {"variant_id": "c77f…", "qty": 1, "unit_price": 1500}
```

-],
- "timestamp": "2025-11-03T12:05:00Z"
- }

24.6 Failover и безопасность

- POS авторизуется через API-токен и подпись SHA256.
- При отсутствии heartbeat > 10 минут: `buy_eligibility → manual_contact`, кнопка «Купить» скрывается.
- POS-запросы идемпотентны (`receipt_number` + `device_id` = уникальный ключ).
- При восстановлении соединения POS синхронизирует все непринятые чеки (bulk upload).
- Ошибки: `409 DUPLICATE_SALE`, `422 INVALID_ITEM`.

24.7 Хранилище и индексы

- **pos_device:** `id`, `shop_id`, `status`, `last_heartbeat_at`, `firmware_version`.
- **pos_sale:** `id`, `shop_id`, `variant_id`, `qty`, `total_price`, `sold_at`.
- **pos_sync_log:** `id`, `device_id`, `status`, `created_at`, `message`.
- Индексы: `btree(shop_id)`, `btree(device_id)`, `btree(sold_at)`.
- Партиционирование по `sold_at`.

25. Система уведомлений (Push / SMS / Email)

25.1 Цели и охват

- Доставлять события платформы пользователям и продавцам: заказы, статусы, чаты, POS-события, модерация, аналитика.
- Каналы: **Push**, **SMS**, **Email**.

- Провайдеры: TBD; предусмотрены интерфейсы `SmsProvider`, `PushProvider`, `EmailProvider` + конфиги.

25.2 Архитектура

- flowchart LR
- SVC[Domain Services (Order, Chat, POS, Moderation)] --> BUS[(Kafka topics: notify.*)]
- BUS --> NOTI[Notifications Worker]
- NOTI --> PREF[(User/Shop Preferences)]
- NOTI --> TMPL[(Template Engine)]
- NOTI --> GATE[Channel Gateways]
- GATE --> PUSH[Push Provider]
- GATE --> SMS[SMS Provider]
- GATE --> MAIL[Email Provider]
- GATE --> LOG[(Delivery Log)]

- **Kafka** топики: `notify.order`, `notify.chat`, `notify.pos`, `notify.moderation`, `notify.analytics`.

- **Notifications Worker** применяет правила предпочтений, дедупликацию, rate limit, ретраи.

- **Template Engine** (Mustache/Handlebars) с i18n и подстановкой переменных.

25.3 События (триггеры)

- **Order:** создан, оплачен (пост-MVP), отменён, готов к самовывозу, доставлен.

- **Chat:** новое сообщение, ответ продавца/покупателя.

- **POS:** оффлайн >10 мин, восстановление, расхождение остатков.

- **Moderation:** пост/товар отклонён/одобрен.

- **Analytics:** суточный отчёт магазину, аномалия (резкий спад/рост).

25.4 Предпочтения и подписки

- Таблица `notification_pref(user_id | shop_id, channel, enabled, quiet_hours, locale, categories[])`.

- Уровни: **пользователь**, **продавец/магазин**, **по категории события**.
- Тихие часы (quiet hours), переадресация на email при оффлайне push.
- Кнопка «Отписаться» для email/SMS (линк с токеном, категория/все).

25.5 Шаблоны и i18n

- Шаблоны: `order_created`, `order_ready_for_pickup`, `chat_new_message`, `pos_offline`, `pos_online`, `moderation_rejected`, `daily_shop_report`.
- Языки: **ru** (активен), **kg/en/zh/kz/uzb** — planned.
- Динамические плейсхолдеры: `{userName}`, `{orderId}`, `{pickupPoint}`, `{shopName}`, `{ts}`, `{ctaUrl}`.

25.6 Дедупликация, ретраи, SLA

- **Idempotency-Key** = хеш(тип+target+payload).
- Дедуп за окно 5 мин по ключу idempotency.
- Ретраи по экспоненте: 1м → 5м → 15м (макс 5 попыток).
- Канальный fallback: Push → Email → SMS (если предпочтения разрешают).
- Отчёт о доставке и статусах в таблице `notification_log` (queued/sent/delivered/failed, provider_ref).

25.7 Безопасность и приватность

- Маскирование PII в логах; хранение токенов пушей/адресов в зашифрованном виде.
- Защита от спама: rate limit per-user/channel (напр., ≤10 SMS/сутки).
- Политики согласия для рассылок (marketing vs transactional).

25.8 API (черновик)

- `GET /notifications/prefs` — получить настройки текущего пользователя/продавца.

- `PUT /notifications/prefs` — изменить настройки (каналы, quiet hours, категории).
- `POST /notifications/test` — отправка тестового сообщения (только owner/admin).
- Webhook для статусов провайдера: `/notifications/webhook/{provider}`.

25.9 Модель данных

- `notification_pref(id, owner_type ENUM[user,shop], owner_id, channel ENUM[push,sms,email], enabled bool, quiet_hours jsonb, locale, categories text[])`
- `notification_template(id, key, channel, locale, subject, body, version, is_active)`
- `notification_log(id, channel, to, template_key, payload_json, status, provider, provider_ref, attempts, last_error, created_at, delivered_at)`

25.10 UI/UX

- В мобильном приложении и на вебе: экран «Уведомления» с тумблерами каналов и категориями.
- Предупреждения о расходах SMS для продавцов (если SMS включены).
- Для критичных событий (POS offline, заказ готов к выдаче) — помечаем как high priority push.
-