Тестовое задание для Python-разработчика

Контекст:
Мы разрабатываем приложение с системой микроплатежей и премиум-контентом.
Нужно создать сервис управления виртуальными товарами и покупками.
Технический стек:

- Python 3.11+
- FastAPI (асинхронный)
- PostgreSQL/SQLite
- Redis (для кэширования)
- Docker + docker-compose

Задача: "Сервис виртуальной экономики"

Часть 1: Модели данных

1. Пользователь (User):
    - id, username, email, balance (int, виртуальная валюта), created_at
2. Виртуальный товар (Product):
    - id, name, description, price (int), type ("consumable" | "permanent"), is_active
3. Инвентарь (Inventory):
    - id, user_id, product_id, quantity (для consumable), purchased_at
4. Транзакция (Transaction):
    - id, user_id, product_id, amount, status ("pending" | "completed" | "failed"), created_at

Часть 2: Бизнес-логика
Реализуйте следующие endpoints:

1. POST /products/{product_id}/purchase — Покупка товара
    - Проверка баланса пользователя
    - Atomic transaction: списание средств + запись в инвентарь + запись транзакции
    - Для consumable товаров увеличивать quantity, для permanent — проверять дубликаты
    - Кэширование: кэшировать инвентарь пользователя на 5 минут
2. POST /users/{user_id}/add-funds — Пополнение баланса
    - Добавить валидацию суммы (> 0, макс. лимит)
    - Idempotency key для предотвращения дублирующих пополнений
3. GET /users/{user_id}/inventory — Получение инвентаря
    - Отдавать из кэша если возможно
    - Группировать consumable товары по quantity
4. POST /products/{product_id}/use — Использование consumable товара
    - Уменьшение quantity в инвентаре
    - Проверка, что товар доступен у пользователя
5. GET /analytics/popular-products — Аналитика популярных товаров
    - Топ-5 товаров по количеству покупок за последние 7 дней
    - Кэшировать на 1 час

Часть 3: Дополнительные требования

1. Фоновая задача:

    - Раз в день сбрасывать кэш инвентаря для всех пользователей
    - Использовать Celery/BackgroundTasks
2. Валидация:
    - Pydantic v2 с кастомными валидаторами
    - Проверка типов товаров, баланса, уникальности операций
3. Обработка ошибок:
    - Кастомные исключения с детализацией
    - Консистентные HTTP статусы

Что будем оценивать особенно строго:

✅ Архитектурные решения — разделение слоев (repository, service, API)
✅ Атомарность операций — особенно в финансовых транзакциях
✅ Эффективность запросов — N+1 проблемы, индексы
✅ Кэширование стратегия — инвалидация, TTL
✅ Idempotency — для финансовых операций
✅ Асинхронность — правильное использование async/await
✅ Тестирование — не только unit, но и интеграционные тесты
Бонусные пункты:

- Написание миграций (Alembic)
- Health-check endpoints
- Rate limiting на чувствительные endpoints
- Простая CI/CD конфигурация

Пример данных для тестирования:

```
json
{
  "products": [
    {
      "name": "Буст на день",
      "type": "consumable",
      "price": 100
    },
    {
      "name": "Премиум-статус",
      "type": "permanent",
      "price": 500
    }
  ]
}
```