**АНАЛИТИКА ПРОЕКТА: СИСТЕМА РАЗБОРКИ/СБОРКИ ТОВАРОВ**

**ЦЕЛЬ ПРОЕКТА**

Создать механизм для разборки товаров-наборов на отдельные части и обратной сборки с сохранением полного жизненного цикла и логирования.

**БИЗНЕС-ПРОБЛЕМА**

* Клиент хочет купить отдельную часть из набора (например, ключ на 8мм из набора ключей)
* Набор нужно разобрать, продать часть, остаток учесть отдельно
* Возможность собрать набор обратно при необходимости
* Учет всех операций в системе

**АРХИТЕКТУРА РЕШЕНИЯ**

**СХЕМА БАЗЫ ДАННЫХ (добавлено)**

* **DisassemblyScenario** - модель сценариев разборки
* **ProductUnit** - расширена полями:
  + disassemblyStatus (MONOLITH, DISASSEMBLED, PARTIAL, COLLECTED, RESTORED)
  + disassembledParentId - связь при разборке
  + disassemblyScenarioId - привязка к сценарию
  + isParsingAlgorithm - флаг возможности разборки
* **ProductUnitPhysicalStatus** - расширен статусами:
  + IN\_DISASSEMBLED - разобранный родитель
  + IN\_COLLECTED - собранные части

**API ROUTES (созданы)**

text

POST /api/disassembly/scenario - создание сценария

POST /api/disassembly/execute - выполнение разборки

POST /api/disassembly/assemble - выполнение сборки

GET /api/disassembly/scenarios - все сценарии

GET /api/disassembly/scenario/[id] - сценарий по ID

GET /api/disassembly/scenario/validate/[id] - валидация

GET /api/disassembly/unit/[unitId]/scenarios - сценарии юнита

**СЕРВИСЫ (созданы)**

* **DisassemblyService** - основная бизнес-логика:
  + createScenario() - создание сценария
  + executeDisassembly() - выполнение разборки
  + executeAssembly() - выполнение сборки
  + validateScenario() - валидация сценария
  + getScenario() / getUnitScenarios() - получение данных

**ФРОНТЕНД КОМПОНЕНТЫ (частично созданы)**

* DisassemblyScenarioForm - форма создания сценария
* DisassemblyExecuteButton - кнопка выполнения разборки
* Страницы: /disassembly, /disassembly/scenario/[id]

**КЛЮЧЕВЫЕ СЦЕНАРИИ**

**1. СОЗДАНИЕ СЦЕНАРИЯ РАЗБОРКИ**

* Выбор юнита-родителя (IN\_STORE, MONOLITH)
* Указание продуктов для создания частей
* Валидация доступности продуктов
* Создание шаблона сценария

**2. ВЫПОЛНЕНИЕ РАЗБОРКИ**

* Проверка валидности сценария
* Создание дочерних юнитов (PARTIAL, IN\_STORE)
* Обновление родителя (DISASSEMBLED, IN\_DISASSEMBLED)
* Логирование операции

**3. ВЫПОЛНЕНИЕ СБОРКИ**

* Поиск доступных частей (PARTIAL, IN\_STORE)
* Обновление статусов частей (COLLECTED, IN\_COLLECTED)
* Восстановление родителя (RESTORED, IN\_STORE)
* Логирование операции

**СТАТУС РАЗРАБОТКИ**

✅ Схема БД и миграции  
✅ Бэкенд сервисы и API routes  
✅ Базовые фронтенд компоненты  
🔄 Интеграция и тестирование  
❌ Расширенные статусы и индикаторы  
❌ Полная фронтенд реализация

**ОСТАВШИЕСЯ ЗАДАЧИ**

1. Создать страницу /disassembly/unit/[unitId]
2. Добавить индикаторы статусов разборки
3. Интегрировать с существующей системой продуктов
4. Протестировать все сценарии
5. Добавить обработку ошибок и валидацию

**ТЕХНИЧЕСКИЕ ДЕТАЛИ**

* Next.js 14 с App Router
* Prisma + PostgreSQL
* Tailwind CSS для стилей
* Полная типизация TypeScript
* Transaction-безопасность операций