



Shopify App Case Study



HIGHLIGHT Bulk Product Editor

PROJECT OVERVIEW



- HIGHLIGHT is a custom-built shopify app designed to simplify large-scale product management.
- The app allows merchants to perform bulk operations on products, variants, pricing, inventory, SEO, and metafields.
- It replicates and enhances core capabilities of leading bulk editors in the shopify ecosystem.
- The system is built for scalability, speed, reliability, and enterprise-level data control.

PROBLEM STATEMENT

- Shopify merchants with large catalogs face operational inefficiencies.
 - Manual editing of hundreds or thousands of products is time-consuming and error-prone.
 - Shopify native admin lacks advanced bulk scheduling, rollback, and preview capabilities.
 - There was a need for a structure bulk-processing engine with tracking and recovery options.



OBJECTIVES OF THE PROJECT



- Develop a high-performance bulk editing engine.
 - Enable price updates using fixed or percentage rules.
 - Provide inventory adjustment (set, add, subtract, overwrite).
 - Support scheduling and rollback of bulk operations.
 - Enable CSV/XLS import and export functionality.
 - Create preview, undo, and activity log mechanisms.

Core Features - Bulk Product Editing



- Bulk update product titles, descriptions, tags, vendors, and product types.
 - Edit SEO fields including meta title and meta description.
 - Update metafields across selected product groups.
 - Apply filters before execution (collection, tag, vendor, etc.).
 - Preview changes before applying them.
 - Undo functionality to restore previous values.

Core Features - Bulk Price Management



Core Features - Bulk Price Management

- Increase or decrease prices by percentage.
 - Apply fixed amount price adjustments.
 - Compare-at-price management support.
 - Scheduled execution for future campaigns.
 - Rollback option to restore original pricing.
 - Batch processing to avoid API rate limits.



**Core Features -
Inventory Management**

Core Features - Inventory Management

- Set inventory to a specific value
 - Increase or decrease stock quantities.
 - Overwrite inventory across locations.
 - Filter products by inventory rules.
 - Process large catalogs using background job queues.



**CSV / XLS Import &
Export System**

CSV / XLS Import & Export System

- Export products data for offline editing.
 - Structured template for bulk upload.
 - Field mapping and validation before execution.
 - Error handling and failure reporting.
 - Safe execution with preview before processing.

Job Queue & Processing Engine



Job Queue &
Processing Engine

- Laravel-based queue system for background execution.
 - Batch processing to handle thousands of records.
 - Progress tracking system in frontend UI.
 - Status indicators: Pending, Running, Completed, Failed.
 - Retry mechanism for failed jobs.

Preview, Undo & Activity Log System



**Preview, Undo &
Activity Log System**

- Snapshot of original product data stored before execution.
 - Preview screen to verify rule impact.
 - Undo feature to revert to previous state.
 - Complete activity logs with timestamps.
 - Merchant-level job history tracking.



**Our Role &
Responsibilities**

Our Role & Responsibilities

- Complete system architecture design.
 - Database schema planning using Laravel migrations.
 - GraphQL Admin API integration.
 - Frontend development using Shopify Polaris.
 - UI development using Shopify Polaris.
 - Backend API, job queues, and validation logic.

Technology Stack



**Technology
Stack**

- Frontend: React.js (dynamic UI & state management).
 - UI Framework: Shopify Polaris
 - Backend : Laravel (API, queue, validation).
 - Database: MySQL with migration-based schema control.
 - API Layer: Shopify Admin GraphQL API.
 - Authentication: Shopify Oauth flow.



**Technical
Architecture**

Technical Architecture

- React frontend communicates with Laravel backend.
- Laravel communicates with Shopify GraphQL API.
 - Bulk operations processes via queued jobs.
 - Database stores job snapshots for rollback.
 - Optimized query batching to respect Shopify rate limits.

Development Process



**Development
Process**

Requirement analysis & competitor benchmarking.

- Feature breakdown and database design.
- GraphQL queries and mutation implementation.
- Queue architecture development.
- Frontend UI creation using Polaris development stores.
- Integration testing with Shopify development stores.
- Performance optimization & deployment.



Major Challenges

- Handling Shopify GraphQL API rate limits.
 - Processing large product datasets efficiently.
 - Maintaining data accuracy during bulk updates.
 - Ensuring rollback restores exact previous values.
 - Keeping frontend responsive during long operations.

Solutions Implemented



- Implemented batched GraphQL mutations.
 - Used Laravel queue workers for background execution.
 - Stored structured JSON snapshots for rollback.
 - Designed optimized filtering queries.
 - Built real-time progress indicators in UI.

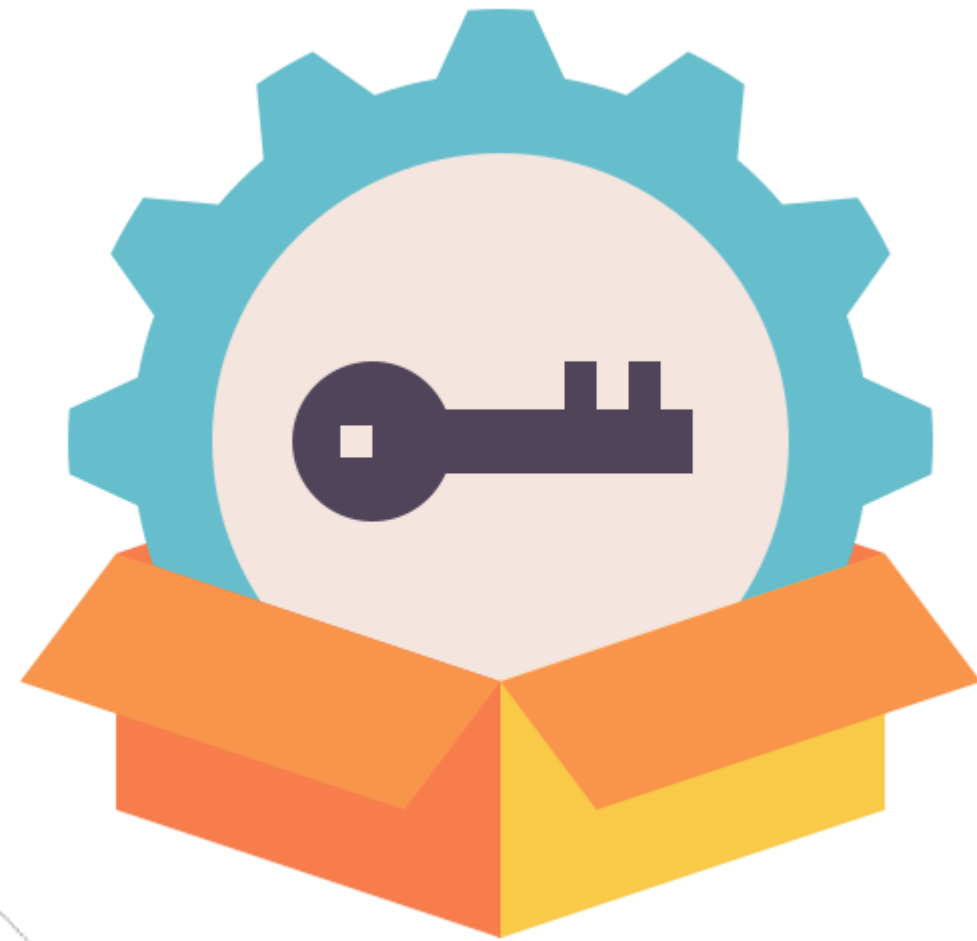
Results & Outcomes

- Implemented batched GraphQL mutations.
 - Used Laravel queue workers for background execution.
 - Stored structured JSON snapshots for rollback.
 - Designed optimized filtering queries.
 - Built real-time progress indicators in UI.



**Results &
Outcomes**

Key Learning



- Deep expertise in Shopify GraphQL API.
 - Advanced understanding of queue-based processing systems.
 - Importance of preview and rollback in bulk operations.
 - Performance optimization in SaaS architecture.
 - Scalable and maintainable full-stack app development

Conclusion



- **HIGHLIGHT** successfully replicates and enhances bulk editing functionality in Shopify.
 - The app is scalable, reliable, and merchant-focused.
 - Built using modern Shopify stack technologies.
 - Demonstrates strong full-stack development capability and SaaS architecture understanding.



**THANK
YOU**