

Product Requirements Document (PRD)

Automated Customer Data Ingestion ETL System

Executive Summary

Product Name: Cloud Native ETL Data Ingestion Platform

Version: MVP 1.0

Team: Engineering Platform Team

Date: September 2025

Status: Requirements Phase

1. Problem Statement

Automotive shops switching to our Shop Management System (SMS) need a reliable way to migrate their existing customer data from competitor systems. Currently, there's no standardized, auditable process to safely ingest customer data while preserving critical relationships between customers, vehicles, invoices, and line items.

Pain Points:

- Manual data migration is error-prone and time-intensive
 - No guaranteed data integrity during transfer
 - Risk of data corruption or loss during migration
 - Inability to rerun failed migrations safely
 - Lack of audit trails for compliance and troubleshooting
-

2. Product Vision & Goals

Primary Goal

Create a serverless, cloud-native ETL system that safely ingests mapped CSV exports from third-party automotive SMS systems into our production MySQL database while preserving data relationships, ensuring auditability, and providing rerun safety.

3. Target Users

Primary Users

- **Customer Support Team:** Manually initiates data transfers during shop onboarding
 - **Partner Integration Team:** Manages third-party data provider relationships
 - **Engineering Team:** Monitors system health and troubleshoots issues
-

4. Product Requirements

4.1 Functional Requirements

Core Data Processing

- **FR-1:** Ingest CSV files for six entity types: Customer, Vehicle, Invoice (Document), Line Item, Payments, Inventory Parts, and Suppliers
- **FR-2:** Preserve parent-child relationships (Customer→Vehicle→Invoice→Line Item and Invoice→Payments)
- **FR-3:** Process all six CSV files (always present) - empty CSV files uploaded when shop doesn't wish to import that data type
- **FR-4:** Handle self-referencing Line Items via externalParentDataLineId
- **FR-5:** Process data in dependency order: Customer→Vehicle→Invoice→Line Item→Payments (Inventory Parts and Suppliers have no order dependency)
- **FR-6:** Differentiate imported invoices from native invoices with visual indicators in UI

File Management & Integrity

- **FR-6:** Accept CSV files via standardized folder structure: `imports/{partner_id}/{shop_id}/{load_id}/`
- **FR-7:** Validate file integrity using manifest.json with SHA256 checksums (post-MVP)
- **FR-8:** Ensure CSV files are transferred before manifest.json to trigger processing
- **FR-9:** Support rerunnable imports without creating duplicates

Data Pipeline

- **FR-10:** Create dynamic staging tables with one-to-one CSV column mapping for all entity types
- **FR-11:** Handle empty CSV files gracefully (skip processing when row count is zero)
- **FR-12:** Perform transactional upserts to production database in dependency order
- **FR-13:** Maintain staging data for 90 days for audit purposes
- **FR-14:** Generate processing ledger with row count validation and transfer summary
- **FR-15:** Create exception reports by file type for failed data transfers (customers, vehicles, invoices, line items, payments, inventory parts, suppliers)

Error Handling & Recovery

- **FR-16:** Provide detailed error logging and failure notifications
 - **FR-17:** Support manual rerun capability for failed jobs
 - **FR-18:** Implement rollback capability for failed transactions
 - **FR-19:** Generate comprehensive exception reports for data validation failures
-

5. Technical Architecture

5.1 System Components

Cloud Infrastructure (GCP)

- **Google Cloud Storage:** File repository with Eventarc triggers
- **Cloud Run Service:** Dispatcher for job orchestration
- **Cloud Run Jobs:** Data processing workers
- **VPC Connector:** Secure database connectivity
- **MySQL Database:** Production data store within VPC

Processing Flow

1. **File Upload:** CSV files + manifest.json uploaded to GCS bucket
2. **Event Trigger:** Manifest.json upload triggers Eventarc
3. **Job Dispatch:** Cloud Run Dispatcher launches processing job
4. **Data Processing:** Cloud Run Job downloads files, stages data, executes upserts in dependency order:
 - Customer → Vehicle → Invoice → Line Item → Payments
 - Inventory Parts and Suppliers processed independently (no order dependency)
5. **Validation:** Row count verification and ledger generation
6. **Exception Reporting:** Generate detailed reports for any failed data transfers by entity type

5.2 Data Contract

File Structure

```
imports/{partner_id}/{shop_id}/{load_id}/
├── customers.csv (always present - required data)
├── vehicles.csv (always present - may be empty)
├── invoices.csv (always present - may be empty, also called documents.csv)
├── line_items.csv (always present - may be empty)
├── payments.csv (always present - may be empty)
├── inventory_parts.csv (always present - may be empty, no order dependency)
├── suppliers.csv (always present - may be empty, no order dependency)
└── manifest.json (trigger file)
```

Manifest Schema

```
json
{
  "load_id": "2025-08-22_partnerA",
  "files": [
    {"name": "customers.csv", "rows": 1234, "sha256": "d6f1a97..."},
    {"name": "vehicles.csv", "rows": 2311, "sha256": "b781f89..."},
    {"name": "invoices.csv", "rows": 856, "sha256": "c892a45..."},
    {"name": "line_items.csv", "rows": 3420, "sha256": "e123b78..."},
    {"name": "payments.csv", "rows": 856, "sha256": "f456c91..."},
    {"name": "inventory_parts.csv", "rows": 0, "sha256": "a789d23..."},
    {"name": "suppliers.csv", "rows": 45, "sha256": "b234e56..."}
  ]
}
```

Note: rows: 0 indicates an empty CSV file when shop chooses not to import that data type

6. MVP Scope & Future Enhancements

MVP Features (Phase 1)

- Manual file transfer by Customer Support team
- Basic CSV ingestion with staging tables for all six entity types
- Transactional upserts to production database in dependency order
- Row count validation and basic ledger
- Exception reporting by entity type for failed transfers
- Visual differentiation of imported invoices in UI
- Error logging and manual rerun capability

Post-MVP Enhancements (Phase 2+)

- **Automated File Transfer:** Direct Dropbox integration
 - **Advanced Validation:** SHA256 checksum verification
 - **Crosswalk Tables:** Deterministic ID mapping for idempotency
 - **Email Notifications:** Automated result reporting
 - **Web UI:** Self-service portal for partners
 - **Advanced Monitoring:** Real-time dashboards and alerts
-

7. User Stories

Customer Support Team







- **US-1:** As a Customer Support rep, I can manually upload CSV files (customers, vehicles, invoices, line items, payments, inventory parts, suppliers) and manifest to initiate data ingestion for a new shop
- **US-2:** As a Customer Support rep, I can view processing status and results in a ledger with detailed row counts for each entity type
- **US-3:** As a Customer Support rep, I can rerun failed ingestion jobs
- **US-4:** As a Customer Support rep, I can review exception reports to understand which specific data failed to transfer for each entity type

Engineering Team





- **US-5:** As an Engineer, I can monitor system health and processing metrics across all entity types
 - **US-6:** As an Engineer, I can troubleshoot failed jobs using comprehensive logs and exception reports
 - **US-7:** As an Engineer, I can verify data integrity through staging table inspection for all six entity types
 - **US-8:** As an Engineer, I can identify imported invoices in the system through visual indicators that differentiate them from native invoices
-

8. Acceptance Criteria




Data Processing

-  Successfully ingest Customer data (always required, never empty)
-  Preserve relationships between Customer→Vehicle→Invoice→Line Item and Invoice→Payments
-  Handle all six CSV files (always present) and process empty files gracefully
-  Process data in correct dependency order while allowing independent processing of inventory parts and suppliers
-  Generate accurate row count validation for all entity types (including zero counts for empty files)
-  Create comprehensive exception reports for failed transfers by entity type

System Reliability

-  Process fails gracefully with detailed error messages
-  Support rerun without creating duplicate records
-  Maintain 90-day staging data retention
-  Complete processing within defined SLA timeframes

Audit & Compliance

-  Generate comprehensive processing ledger
 -  Maintain complete audit trail of all operations
 -  Provide error logs for troubleshooting
-

9. Risk Assessment

High Priority Risks

- **Data Loss Risk:** Mitigation through staging tables and transactional processing
- **Performance Degradation:** Mitigation through Cloud Run auto-scaling
- **Security Breach:** Mitigation through VPC isolation and encryption

Medium Priority Risks

- **Third-Party Dependencies:** Standardized data contracts and validation
 - **Manual Process Errors:** Clear documentation and eventual automation
-

10. Timeline & Milestones

Phase 1 - MVP (8-12 weeks)

- **Weeks 1-2:** Infrastructure setup and basic pipeline
- **Weeks 3-4:** CSV processing and staging table creation
- **Weeks 5-6:** Production database upsert logic
- **Weeks 7-8:** Testing, validation, and error handling
- **Weeks 9-12:** Documentation, deployment, and team training

Phase 2 - Enhancements (Future)

- Automation features
 - Advanced validation
 - Web UI development
 - Monitoring and alerting improvements
-

11. Dependencies & Assumptions

Dependencies

- GCP account setup and permissions
- MySQL production database access
- Third-party data provider cooperation
- Customer Support team training

Assumptions

- Standardized CSV format compliance from partners
- Stable network connectivity for cloud services
- Adequate GCP resource quotas
- Team availability for testing and validation

This PRD serves as the foundation for MVP development and will be updated based on stakeholder feedback and technical discoveries during implementation.