Software Design Document

Version 1.0

Date: 2/6//2026
Team Name: Evergreen Systems
Project Sponsor: Kyle Montgomery
Faculty Mentor: Nazmul Hossain
Team Members:

- (Team Lead) Asher Romanenghi
- Mark Johnson
- Tyler Sturm
- Melvin Agram

Overview: This Software Design Specification document defines the architecture and technical blueprint for the Generous Commerce Connectors project. It outlines the system design, major components, implementation strategy, and key constraints that will guide development and final delivery.

Table of Contents

1. Introduction

The gift-giving industry is a massive and continually evolving segment of ecommerce, valued at over $150 billion annually in the United States alone. Despite its scale, the process of connecting gift buyers, brands, and fulfillment partners remains fragmented. Merchants often rely on multiple sales platforms to reach customers, leading to data inconsistencies and time-consuming manual catalog management. As consumers expect increasingly personalized and frictionless shopping experiences, efficient integration between retailers and digital marketplaces has become essential.

To address these challenges, Evergreen Systems is developing a series of ecommerce platform integration apps that automates catalog ingestion and transformation into Generous's unified data model. This project will deliver platform-specific connectors that allow SFCC, Shopify, and other platform merchants to connect directly with Generous's platform. By simplifying catalog synchronization and eliminating redundant data conversion, this solution aims to reduce onboarding time, improve data accuracy, and establish a scalable model for future commerce integrations.

2. Implementation Overview

We are building a multi-platform catalog ingestion system that allows merchants to connect their commerce platform, publish selected products/collections with required gifting metadata, and keep that catalog synchronized in Generous via a single canonical product model. The design is driven by the core challenge that each platform represents catalog data differently, while merchants expect a low-friction setup inside their existing admin tools and clear, actionable feedback when sync issues occur.

This implementation must fit within Generous's existing backend, deployment, security, and monitoring standards, and it must comply with platform-specific constraints such as Shopify Admin app patterns and SFCC Business Manager extension lifecycles and auth models. It also needs to handle order-of-magnitude variability in catalog size and submission volume, provide timely status and error visibility (even when ingestion is asynchronous), and remain resilient to partial failures and transient platform/API issues without corrupting sync state.

Our approach separates platform-specific and platform-agnostic concerns: native admin plugins manage merchant connection, scoped authorization, and configuration, while a versioned ingestion API defines a stable submission contract for full refreshes and incremental updates. Behind that contract, a queue-backed processing layer performs server-side validation and normalization into Generous's canonical product representation, persisting both catalog state and per-run sync metadata so merchants can see progress, item-level errors, and outcomes, and so the

system can retry safely and support future connectors without changing the core ingestion pipeline.

3. Architectural Overview

The Generous Commerce Connectors architecture is designed to handle the inherent complexity of the gift-giving industry by centralizing fragmented data from various merchants. The system follows a **decentralized, modular architectural style** that separates platform-specific concerns at the "edges" while maintaining a robust, unified core for data processing.

3.1. High-Level Architecture

Connecting Generous backend to SFCC (Salesforce Commerce Cloud) and other E-commerce platforms (Shopify, Zapier, Adobe, etc). There are four distinct stages to ensure that product data moves in a precise, predictable, secure, fashion so that it can be ready for Generous's AI agent.

- **1. The Stores** (Source Layer)**:** This is where the process begins. Native merchant plugins, such as the **Salesforce Cartridge** or Shopify Admin apps, live directly within the merchant's existing environment. Their primary responsibility is to act as data exporters, allowing merchants to "push" their catalogs and select specific products for gifting.

- 2. **The Security Gate** (API & Auth): Once data is pushed, it hits the **OAuth 2.0** security layer. This serves as the "handshake" that ensures data isolation, ensuring one merchant's catalog remains strictly separate and protected from others.

- **3. The Factory** (Backend Processing): This is the engine room of the architecture. Because catalogs can contain thousands of items, the system uses an **asynchronous job processing model**.
    - **Background Workers** pick up tasks from a queue so the merchant's interface stays fast.
    - The **Data Cleaner (Transformer)** then normalizes the messy, platform-specific data into the **Universal Product Schema**.

- **4. The Library (Database)**: Finally, the standardized data is saved to a **MongoDB** database. This flexible "library" stores not just the products, but the entire history of ingestion jobs and sync metadata.

3.2. Key Communication Mechanisms

- **Asynchronous Ingestion:** Merchants initiate a “Sync” from their Salesforce Business Manager. The Backend processes these large catalogs as background jobs to ensure scalability and system responsiveness.
- **OAuth 2.0 Security:** This provides the “handshake” that ensures data isolation between different merchants and prevents unauthorized API access.
- **Real-time Verification:** Live API calls to the Salesforce Cartridge to verify up-to-the-minute **inventory status and pricing**.

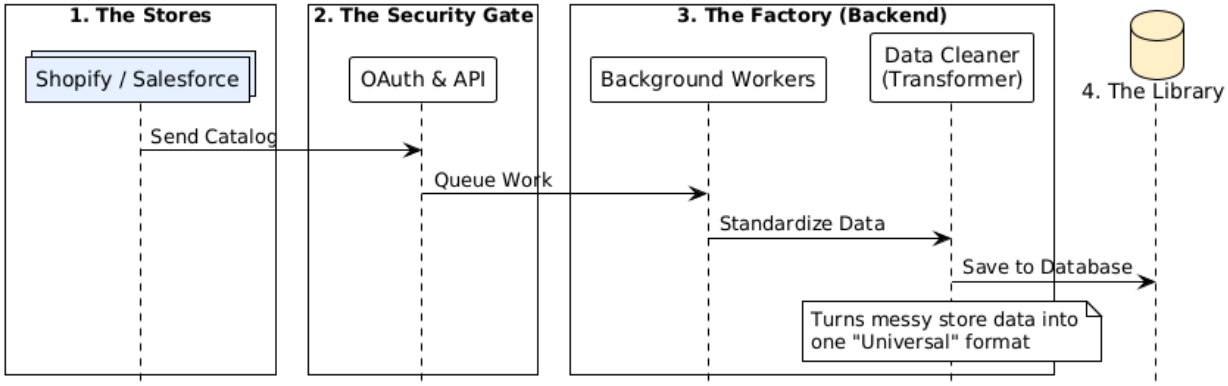


Figure 1. High Level Architecture of Generous Commerce Connectors

3.3. Architectural Style Commentary

The architecture employs a **Versioned API-First approach**. By defining a stable "canonical" product model in the center, Evergreen Systems ensures that adding a new connector—such as Adobe Commerce or Zapier—does not require changing the core ingestion pipeline. This separation of concerns keeps the platform-native tools "lightweight" and the central backend "intelligent" and scalable.

4. Component-Level Design:

At a high level, platform specific tools collect product data, and the Generous backend handles processing, validation, and storage. The system moves step by step from connectors to the API then to background processing and finally to the database.

The system is made up of four main components:

- Store/Platform Connectors

- RESTful API and Auth
- Validation and Normalization
- Library/Database Layer

4.1. Store Platforms

The platform connectors are small apps that run inside Salesforce Commerce Cloud. They allow merchants to connect their store and send product catalogs to Generous.

Their responsibilities are simple. They:

- authenticate the merchant
- let the merchant choose products or collections
- export catalog data
- send the data to the Generous API
- show sync status or errors

They only collect and send data. They do not validate, process, or store anything. This keeps the connectors lightweight and easy to update.

4.2. API and authentication

The API is the main entry point into the Generous system. All connectors communicate with this service. It processes all the jobs at once in a queue system to prevent any jobs from timing out or being blocked. Once the job is scheduled it is authenticated and allowed into the generous database.

Its job is to:

- verify data transmission

- accept catalog submissions
- Connect to generous database

The API does not do heavy processing itself. Instead it quickly hands work off to background workers so the system stays fast and responsive.

4.3. Validation and Normalization

Different platforms format their product data differently. This component makes sure all data follows one common structure before it is saved.

It:

- checks required fields
- cleans incorrect values
- standardizes formats
- converts data to the Generous product schema

This ensures that the rest of the system always works with consistent data models, allowing generous to work with one default schema.

4.4. Library Layer

The database layer stores all system data permanently. MongoDB is used because it works well with flexible JSON data from many different platforms. Mongoose adds structure and validation.

It stores:

- products
- catalogs
- ingestion jobs
- sync history

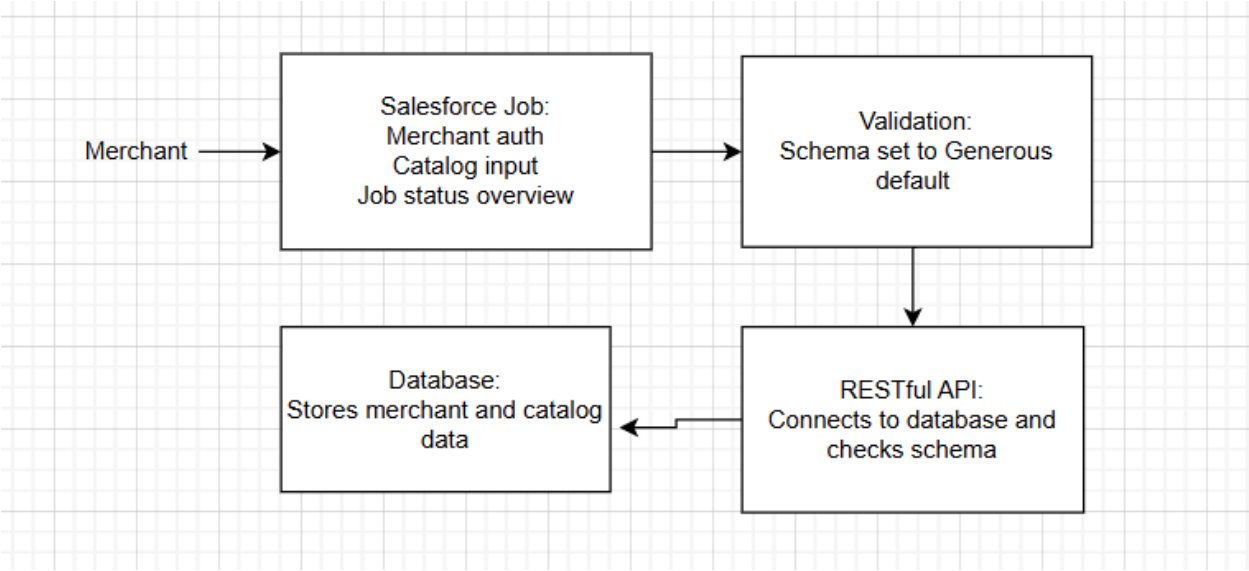


Figure 2. Component Design of Generous Data Flow

5. Implementation Plan

This implementation plan translates the system design into a realistic execution strategy for delivering the Generous Commerce Connectors platform. It outlines the development phases, implementation order of system components, integration strategy, and incremental delivery approach across three planned sprints. The objective is to ensure the architecture is implemented in a structured and testable manner while allowing for parallel development and continuous validation with the project sponsor.

The system will be developed iteratively, beginning with core catalog ingestion functionality and expanding into multi-platform support and full system integration. Each sprint concludes with a working milestone that can be demonstrated and validated.

Implementation Timeline and Development Phases -

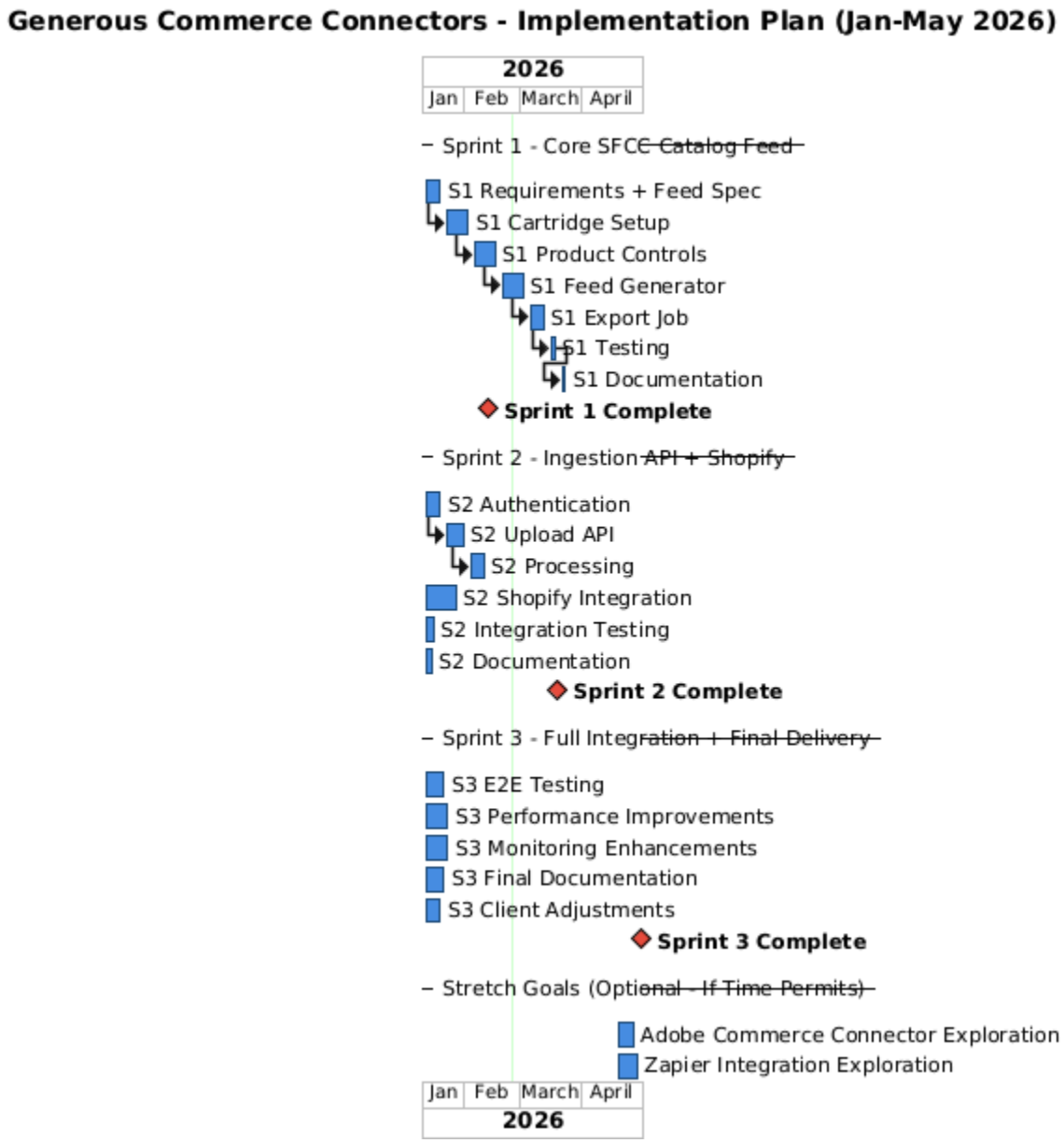
The project follows a three-sprint implementation model aligned with capstone deliverable deadlines:

- **Sprint 1 (Due February 13, 2026):** Core Salesforce Commerce Cloud (SFCC) catalog feed and cartridge implementation

- **Sprint 2 (Due March 20, 2026):** Secure ingestion API, job processing pipeline, and Shopify integration

- **Sprint 3 (Due May 1, 2026):** Full system integration, testing, monitoring, and final delivery

The graphical timeline below illustrates the sequence of component implementation, testing, and integration across all phases while highlighting opportunities for parallel work.



During Sprint 1, development focuses on building the foundational product feed pipeline. This includes extracting product data from Salesforce Commerce Cloud (SFCC), mapping catalog data to the Generous schema, generating structured JSON feeds, and implementing Business Manager controls that allow merchants to enable or disable products for export. A scheduled export job and initial documentation are also completed. These components establish the baseline ingestion workflow and validate the core system architecture.

Sprint 2 expands the system to support scalable ingestion and multiplatform connectivity. Development includes implementing secure merchant authentication, feed upload endpoints, and an asynchronous job processing system for handling catalog uploads. Shopify integration is developed in parallel with ingestion APIs to support multiplatform catalog ingestion, with team members working concurrently on different platform components. Logging, error handling, and job status tracking are introduced to improve system reliability and visibility.

Sprint 3 focuses on full system integration and production readiness. End to end testing across SFCC, Shopify, and the ingestion pipeline validates data accuracy and stability. Performance improvements, monitoring features, application packaging, and final documentation are completed, followed by client validation and final delivery. If core functionality is completed ahead of schedule, stretch goals will be pursued. These include exploring additional commerce platform integrations such as Adobe Commerce and Zapier to further extend the connector architecture.

System components are implemented in dependency order to reduce risk. Product feed generation and schema mapping are developed first, followed by ingestion APIs and job processing services. Platform connectors are integrated once the ingestion pipeline is stable. Each subsystem is tested independently before full end to end integration, allowing early detection of mapping errors, authentication issues, and schema mismatches. Parallel development is supported by dividing work across connectors, backend APIs, documentation, and testing, with stretch goal work introduced only after core requirements are stable and validated.

5.1. Assumptions, Constraints, and Risks

Implementation assumes continued access to Salesforce Commerce Cloud and Shopify sandbox environments, stable product feed specifications, and timely feedback from the project sponsor throughout development. The team also assumes that merchant catalog data will remain consistent with documented schema requirements and that required API credentials and permissions will be available when needed for testing and validation.

Key constraints include limited development time within the academic semester, evolving external APIs and platform documentation, and the need to maintain compatibility across multiple ecommerce systems. Coordination with external platforms and sandbox environments may introduce delays that are outside of the team's direct control. These constraints require prioritizing core catalog ingestion functionality and ensuring that required deliverables are completed before optional enhancements or stretch goals are pursued.

Technical risks include schema mismatches between merchant data and the Generous product schema, platform specific development complexity associated with Salesforce Commerce Cloud cartridge architecture, authentication and security concerns related to API access, and scalability challenges when processing large product catalogs. Additional risk exists in coordinating multi platform integration work across team members while maintaining consistent data mapping and system behavior.

These risks are mitigated through early schema validation, modular connector design, consistent use of shared data mapping standards, secure authentication practices, and implementation of asynchronous job processing for catalog ingestion. Regular testing with sample merchant data and continuous client feedback further reduce integration risk. Trade offs prioritize delivery of a stable and scalable ingestion pipeline over implementation of additional platform integrations or advanced features, ensuring reliable core functionality before expansion through optional stretch goals such as additional commerce platform connectors.

6. Conclusion

The Generous Commerce Connectors project defines a scalable, platform-aligned approach to solving Generous's primary onboarding bottleneck: reliably importing and maintaining merchant catalogs without manual intervention. By combining thin, platform-native merchant plugins (Shopify Admin and SFCC Business Manager) with a single, versioned catalog submission API, the design keeps platform-specific concerns at the edges while centralizing validation, normalization, persistence, and observability in Generous-owned services.

This architecture supports the core requirements of secure merchant authorization, selective publishing of products/collections, clear sync feedback, and resilient ingestion across heterogeneous schemas. The asynchronous job processing pipeline enables predictable performance for catalogs of widely varying sizes, while idempotent sync semantics and structured validation results reduce operational risk and simplify retries. The phased implementation plan further de-risks delivery by establishing the SFCC export baseline first, then expanding into a hardened ingestion pipeline and Shopify integration, and finally completing end-to-end testing, monitoring, and production readiness.

Taken together, this design provides a durable foundation for current connectors and a repeatable pattern for future platform integrations, enabling Generous to scale merchant onboarding and maintain high-quality product data for AI-driven discovery and gifting workflows.