G1 PO App - Developer Handover Report

Last Updated: May 31, 2025

1. Project Overview

The "G1 PO App" is a web application designed to streamline order processing for Global One Technology. It ingests orders from BigCommerce, allows users to manage these orders, generate purchase orders (POs) to suppliers, create shipping labels (UPS and FedEx), generate packing slips, and manage related international shipment documentation. The application features a Flask backend (Python) and a React frontend (Vite). Authentication is handled by Firebase.

2. Project Layout & Core Technologies

- **Frontend:**
    - Framework: React (using Vite for build tooling)
    - Key Components: Dashboard.jsx, OrderDetail.jsx, InternationalOrderProcessor.jsx, DomesticOrderProcessor.jsx, Login.jsx, App.jsx (routing), AuthContext.jsx (Firebase auth).
    - Styling: CSS (e.g., Dashboard.css, OrderDetail.css)
- **Backend:**
    - Framework: Flask (Python)
    - Main file: app.py
    - Blueprints: orders.py, international.py, suppliers.py, hpe_mappings.py, quickbooks.py, reports.py, utils_routes.py for modular routing.
    - Key Services:
        - shipping_service.py: Handles interactions with UPS and FedEx APIs for label generation and shipment processing.
        - document_generator.py: Generates PDF documents like Purchase Orders and Packing Slips using ReportLab.
        - email_service.py: Sends emails (e.g., POs to suppliers, notifications) via Postmark.
        - gcs_service.py: Manages file uploads/downloads with Google Cloud Storage (for documents like labels, POs, logos).

        - iif_generator.py: (Likely for QuickBooks IIF file generation, though not extensively covered in recent interactions).
- **Database:**
    - PostgreSQL (likely hosted on Google Cloud SQL, given GCP context).
    - SQLAlchemy is used for database interactions in the Flask backend.
- **Authentication:**
    - Firebase Authentication (for user login and protecting backend routes).
- **Deployment (Assumed):**
    - Backend (Flask): Google Cloud Run (containerized with Docker).
    - Frontend (React/Vite): Firebase Hosting (common for Vite/React apps) or another static site hosting solution.
- **Cloud Services:**
    - Google Cloud Platform (GCP):
        - Cloud Run (for backend hosting).
        - Cloud SQL (for PostgreSQL database).
        - Google Cloud Storage (GCS) (for storing generated documents, logos).
        - Google Container Registry (GCR) or Artifact Registry (for Docker images).
    - Firebase: Authentication, potentially Hosting.
- **Third-Party APIs:**
    - BigCommerce API (for order ingestion, status updates).
    - UPS API (for shipping labels, international documentation).
    - FedEx API (for shipping labels).
    - Postmark API (for sending emails).
    - BrokerBin (links generated for part lookups).

3. Key Features Implemented & Steps Taken (Summary)

- **User Authentication:** Firebase-based login and protected routes.

- **Order Ingestion:** Pulls orders from BigCommerce (via /ingest_orders endpoint in orders.py).
    - Parses customer messages for freight details (carrier, service, account numbers, ZIP for UPS).
    - Parses customer messages for compliance IDs.
    - Determines if an order is international.
    - **Recent Change:** All new orders (domestic and international) are now ingested with status "new" (removed "international_manual" status).
- **Dashboard (Dashboard.jsx):**
    - Displays orders with filtering by status.
    - Shows status counts.
    - Button to trigger order ingestion.
    - Navigation to "Daily Sales" report.
    - **Recent Change:** Removed "International" / "international_manual" from status filter and related highlighting logic.
- **Order Detail View (OrderDetail.jsx):**
    - Displays detailed order information, line items.
    - Links to BrokerBin for part numbers.
    - Allows manual status updates (e.g., to "RFQ Sent", "Pending", "Completed Offline").
    - Conditionally renders either DomesticOrderProcessor or InternationalOrderProcessor based on order.is_international.
    - Displays calculated profit for processed orders.
    - **Recent Change:** The formatShippingMethod helper function was significantly refined to correctly display "UPS Standard" (from "UPS® (UPS Standard℠)") and differentiate it from "FedEx Standard Overnight" and other UPS services, ensuring the "Ship via:" display is accurate.
- **Domestic Order Processing (DomesticOrderProcessor.jsx):**

    - Handles single supplier POs, multi-supplier POs, and G1 Onsite Fulfillment for domestic orders.
    - Generates PO PDFs, Packing Slips.
    - Integrates with UPS and FedEx APIs for label generation via shipping_service.py.
    - Supports "Bill Sender", "Bill Recipient (UPS/FedEx)", and "Bill Third Party (UPS)".
    - Real-time profit calculation based on form inputs.
    - SKU description lookup.
- **International Order Processing (InternationalOrderProcessor.jsx):**
    - Fetches international-specific details (customs info, compliance fields) via /order/<id>/international-details (international.py).
    - Allows selection of fulfillment strategy (Supplier PO or G1 Onsite International).
    - Handles PO creation if a supplier is chosen.
    - **UPS International Shipments:**
        - Constructs detailed payload for UPS API.
        - Generates international shipping labels and customs forms (Commercial Invoice) via shipping_service.py.
        - **Bill Sender:** Default payment method.
        - **Bill Receiver (UPS):** Implemented. Requires recipient's UPS account number and the postal code associated with that account. The UI defaults the postal code to the customer's billing address ZIP. PaymentInformation in the UPS payload is correctly set to BillReceiver with AccountNumber and Address (containing PostalCode and CountryCode).
        - **InvoiceLineTotal (for Canada & Puerto Rico):** Logic added to calculate the total invoice value. If the destination is CA or PR and the value is >= $1.00, InvoiceLineTotal (with CurrencyCode and MonetaryValue) is added directly under the Shipment object. For

other international destinations, it's added under Shipment.ShipmentServiceOptions.InternationalForms.

        - **Overall Shipment Description:** Logic implemented to generate a more descriptive summary if multiple line items with different customs descriptions exist. It strips common suffixes (e.g., " for computers"), joins the unique parts, re-appends the suffix if space allows (within 50 chars), or uses a generic term ("Computer Components").
        - **Shipping Service Dropdown Default:** Defaults to the customer's selected shipping method from BigCommerce by mapping the BigCommerce shipping string (e.g., "UPS® (UPS Standard℠)") to the corresponding UPS service code (e.g., "11").
        - **Length, Width, Height Fields:** Removed from the UI as they are not strictly required for basic UPS international label generation if not rating.

- **Document Generation (document_generator.py):**
    - Generates PO PDFs and Packing Slip PDFs using ReportLab.
    - Supports custom fonts (Eloquia, falling back to Helvetica).
    - Handles blind shipping for packing slips.
    - Logo integration from GCS (supports SVG via svglib if installed, and PNG/JPG via Pillow).
    - **Recent Change:** Payment method display on packing slips now strips bracketed content (e.g., "Net 30 Terms [with credit approval]" becomes "Net 30 Terms").
- **Shipping Services (shipping_service.py):**
    - Handles OAuth token generation for UPS and FedEx.
    - generate_ups_label_raw(): Generates domestic UPS labels, supports Bill Sender and Bill Third Party (using PaymentInformation with BillThirdParty and ShipperNumber).
    - generate_fedex_label_raw(): Generates FedEx labels, supports Bill Sender and Bill Recipient.

    - generate_ups_international_shipment(): Takes a fully formed payload from the frontend, makes the UPS API call, and processes the response for labels and forms. Converts GIF labels to PDF.
        - **Recent Change:** Logic added to transform full Canadian province names (e.g., "Ontario") received from the frontend into two-letter abbreviations (e.g., "ON") for ShipTo.Address.StateProvinceCode and SoldTo.Address.StateProvinceCode before sending to UPS.
- **Email Service (email_service.py):**
    - Sends emails using Postmark.
    - Sends POs to suppliers with attachments (PO PDF, label, packing slip).
    - Sends internal sales notifications for G1 Onsite fulfillments.
    - **Recent Change:** Added send_debug_email function to send debug information (including payloads as attachments) on processing errors.
- **Backend API Endpoints (app.py, various blueprints):**
    - CRUD operations for Suppliers and HPE Descriptions.
    - Lookup endpoints (spare parts, descriptions).
    - QuickBooks sync functionality.
    - Daily revenue reports.
- **Error Handling & Logging:**
    - Basic error handling in frontend components (setProcessError).
    - Extensive DEBUG and INFO print statements in backend Python files, which act as logs in Cloud Run.
    - Transaction rollbacks in backend routes on error.

4. Docker & Deployment Workflow (Based on User Interactions)

- The backend is containerized using Docker.
- A Dockerfile exists in the backend's root directory.
- The build command used by the user: docker build -t gcr.io/<PROJECT_ID>/g1-po-app-backend-service:<TAG> .

- The image is intended to be pushed to Google Container Registry (GCR). docker push gcr.io/<PROJECT_ID>/g1-po-app-backend-service:<TAG>
- This image is then deployed to Cloud Run using: gcloud run deploy <SERVICE_NAME> --image gcr.io/<PROJECT_ID>/g1-po-app-backend-service:<TAG> --region <REGION> --project <PROJECT_ID> (Or using --source . for source-based deployments if preferred).

5. Recent Key Changes & Fixes Summary (from this chat session)

- **International BillReceiver (UPS):**
    - Implemented functionality for billing international UPS shipments to the recipient's account.
    - InternationalOrderProcessor.jsx now constructs the PaymentInformation.BillReceiver object with AccountNumber and Address: {PostalCode, CountryCode}. The Postal Code defaults to the customer's billing ZIP and the Country Code to the recipient's shipping country.
- **Overall Shipment Description (International):**
    - Enhanced logic in InternationalOrderProcessor.jsx to create a more representative description for multi-item international shipments, stripping common suffixes and concatenating unique descriptions within the 50-character limit.
- **UPS Standard Service:**
    - Added "UPS Standard" (service code "11") to SHIPPING_METHODS_OPTIONS_INTL in InternationalOrderProcessor.jsx.
    - Updated mapBcShippingToIntlServiceCode in InternationalOrderProcessor.jsx to correctly map "UPS® (UPS Standard℠)" to code "11".
    - Corrected formatShippingMethod in OrderDetail.jsx to accurately display "UPS Standard" and avoid conflict with "FedEx Standard Overnight" by removing the overly broad "nda" check for Next Day Air.
- **InvoiceLineTotal for Canada/PR (UPS International):**
    - Logic added to InternationalOrderProcessor.jsx to calculate and include InvoiceLineTotal (if value >= $1.00) directly under the Shipment object for US

to CA/PR shipments, or under InternationalForms for other destinations, as per UPS requirements.

- **Removal of LWH Fields (International):**
    - Length, Width, and Height input fields and their corresponding states were removed from InternationalOrderProcessor.jsx. The Package.Dimensions object is still conditionally added to the UPS payload if these values were somehow provided (though the UI path is removed).
- **Removal of "international_manual" Status:**
    - orders.py: ingest_orders_route now sets all new orders (domestic and international) to "new". "international_manual" removed from finalized_or_manual_statuses and allowed_statuses for status updates.
    - Dashboard.jsx: "International" filter option removed from orderedDropdownStatuses. hasPendingOrInternational state changed to hasPendingOrders and logic updated.
    - OrderDetail.jsx: Conditional rendering for the "Mark as Completed Offline" button updated to only check for "pending" status.
- **Packing Slip Payment Method Display:**
    - document_generator.py: Added _format_payment_method_for_packing_slip to strip bracketed content from payment method strings (e.g., "Net 30 Terms [with credit approval]" becomes "Net 30 Terms").
- **Debug Email for Failed Shipments:**
    - email_service.py: Added send_debug_email function.
    - international.py (process_international_dropship_route): Implemented logic to email the UPS request payload as a text attachment upon shipment processing failure.
- **Canadian Province Code Conversion (International):**
    - shipping_service.py (generate_ups_international_shipment): Added logic to convert full province names for Canada (e.g., "Ontario") to their two-letter abbreviations (e.g., "ON") for ShipTo and SoldTo addresses before sending to the UPS API.

- **Local Development Setup Guidance:** Provided general instructions for setting up Vite (frontend) and Flask (backend) servers locally, including virtual environments, dependencies, environment variables (.env), Vite proxy, and Firebase Admin SDK setup.
- **Docker Command Correction:** Corrected docker build command typos. Identified that Docker Desktop not running was the cause of connection errors.

6. Remaining Tasks & Future Considerations (High-Level)

- **Thorough Testing of All Scenarios:**
    - Domestic: Single Supplier, Multi-Supplier, G1 Onsite (Bill Sender, Bill Recipient UPS/FedEx, Bill Third Party UPS).
    - International: Supplier PO, G1 Onsite (Bill Sender, Bill Receiver UPS - especially with the new postal code requirement for BillReceiver.Address).
    - All document generation (POs, Packing Slips for various scenarios including blind).
    - All email notifications.
    - Edge cases for InvoiceLineTotal (e.g., $0 value shipments to CA/PR, if possible).
    - Various shipping method mappings from BigCommerce.
- **FedEx International Shipments:**
    - Currently, InternationalOrderProcessor.jsx is primarily focused on UPS. If FedEx international is needed, a similar level of detail for payload construction, service code mapping, and handling different billing types (Bill Sender, Bill Receiver, Bill Third Party - if supported and needed) will be required.
    - shipping_service.py would need a generate_fedex_international_shipment function, analogous to the UPS one.
- **"BillThirdParty" for UPS International:**
    - The UI in InternationalOrderProcessor.jsx still has the customerUpsAccountZipCode field. The logic for PaymentInformation currently uses this (along with account number and country code) for BillReceiver.Address. If a distinct "BillThirdParty" (where the payer is neither

shipper nor receiver) is needed, the UI might need a way to distinguish this and collect third-party specific address/postal code details if they differ from the receiver's. The current implementation effectively uses receiver details for the payer's address in BillReceiver.

- **UI/UX Refinements:**
    - Consistent error display and feedback to the user.
    - Loading indicators for all async operations.
    - Potentially more granular display of processed PO/shipment information in OrderDetail.jsx after processing.
- **Refined Profitability Calculation (International):**
    - The current profit calculation for international in InternationalOrderProcessor.jsx is basic (sets cost to 0 for G1 Onsite). This may need to incorporate actual shipping costs, duties, taxes, or other international fees if they become available or can be estimated.
- **Configuration Management:**
    - Review all environment variables and ensure they are clearly documented and managed securely, especially for production.
- **Code Cleanup & Optimization:**
    - Refactor shared helper functions (like formatShippingMethod, mapBcShipping...) into a utils.js file in the frontend if they are used in multiple components.
    - Review Python backend for any redundant code or areas for optimization.
- **Comprehensive Logging & Monitoring (Production):**
    - While print statements work for Cloud Run logs, consider integrating a more structured logging library in Python for better log management and analysis in production.
    - Set up monitoring and alerting for critical errors.
- **Database Schema Review:**

    - Consider if additional fields are needed in shipments or purchase_orders tables to store more details about billing (e.g., payment_type_code, payer_account_number, payer_postal_code).
- **Security Hardening:** Review all API endpoints, authentication/authorization logic, and input validation.
- **QuickBooks Integration (iif_generator.py):** Further development and testing if this is a core requirement.

7. Technical Details for Developer

- **Backend Entry Point:** app.py (runs Flask development server or is used by Gunicorn in production).
- **Frontend Entry Point:** src/main.jsx (for Vite/React).
- **Key Data Flow for Order Processing:**
    1. Dashboard.jsx -> /api/ingest_orders (orders.py) -> BigCommerce API -> orders table.
    2. Dashboard.jsx -> /api/orders (orders.py) -> Displays orders.
    3. User navigates to OrderDetail.jsx (/orders/:orderId).
    4. OrderDetail.jsx fetches data via /api/orders/:orderId (orders.py) and /api/suppliers (suppliers.py).
    5. OrderDetail.jsx renders DomesticOrderProcessor.jsx or InternationalOrderProcessor.jsx.
    6. Processor components:
        - InternationalOrderProcessor.jsx fetches additional data via /api/order/:orderId/international-details (international.py).
        - User fills forms.
        - On submit, InternationalOrderProcessor.jsx calls /api/order/:orderId/process-international-dropship (international.py).
        - DomesticOrderProcessor.jsx calls /api/orders/:orderId/process (orders.py).
    7. Backend processing routes (process_order_route in orders.py or process_international_dropship_route in international.py):

        - Interact with shipping_service.py for labels.
        - Interact with document_generator.py for PDFs.
        - Interact with gcs_service.py for uploads.
        - Interact with email_service.py for notifications.
        - Update database tables (purchase_orders, po_line_items, shipments, orders status).
        - Update BigCommerce (create shipment, update status) via shipping_service.py helper functions.
- **Environment Variables:** Critical for API keys, database connections, etc. Ensure .env is correctly set up locally and corresponding environment variables are set in Cloud Run. See app.py for a list of environment variables used.
- **UPS API Version:** Currently set to v2409. Check if this is the latest or most appropriate for the features being used.
- **State Management (Frontend):** Primarily component-level state using useState and useEffect. AuthContext for global user/API service.
- **Backend Database Interactions:** Uses SQLAlchemy Core (text-based queries). Transactions are used for atomic operations.
- **Error Codes/Messages:**
    - UPS: "120437" for missing/invalid postal code of payment account.
    - Custom error messages are returned from API endpoints.

This report should provide a solid foundation for the next developer to understand the project and continue its development.