Project Handover Report: G1 PO App - International Shipping Module Enhancement

Date of Report: May 27, 2025 (Updated)

Compiled By: Gemini Assistant (incorporating original report by Gemini Assistant dated May 24, 2025, and subsequent interactive development session)

Project: G1 PO App (Purchase Order and Shipment Automation)

Table of Contents:

---

1. Project Goal & Core Value Proposition

(As per original Handover Report) The primary goal is to develop the "G1 PO App," a secure, internal web application to automate and streamline the purchase order (PO) and shipment processes for both drop-shipped and G1-stocked items. This application aims to minimize manual effort, enhance accuracy, serve as a central platform for critical business workflows, and improve overall operational efficiency. Key functionalities include ingesting orders from BigCommerce, managing orders through a web interface, generating POs, packing slips, and shipping labels (UPS & FedEx), facilitating supplier communication, updating BigCommerce with shipment details, and integrating with QuickBooks Desktop via IIF files.

---

2. Current State of the Application (Updated)

The application has a significant number of core features implemented and has undergone a major backend refactoring to use Flask Blueprints. The refactored backend is successfully deployed to Google Cloud Run.

Key Achievements & Functionalities (Including Recent International Shipping Module Progress):

- **Order Management**: Robust BigCommerce order ingestion and management.
    - The ingest_orders_route in orders.py has been updated to parse international compliance IDs from order comments and store them in a new compliance_info JSONB column in the orders table.
- **Fulfillment Modes**:
    - Supports Single/Multi-Supplier POs and G1 Onsite Fulfillment for domestic orders.
    - International order processing for drop-ship scenarios (PO generation + international label) and G1 direct international shipments is now being built out in the UI and backend structure.
- **Document Generation**: PDF generation for Purchase Orders and Packing Slips, including a "blind drop ship" version.
- **Shipping Integration (Existing Domestic)**:
    - **UPS**: "Bill Sender" and "Bill Third Party" label generation is functional for domestic shipments.
    - **FedEx**: OAuth working; "Bill SENDER" labels tested. "Bill RECIPIENT" blocked by FedEx account configuration.
- **International Shipping Data Capture (New)**:
    - **BigCommerce Checkout**: Custom JavaScript (custom-checkout.js loaded via BigCommerce Script Manager) dynamically adds input fields for international compliance IDs (EORI, VAT, IOSS, etc.) based on the selected shipping country. These IDs are appended to the order comments.
    - **Backend API for Frontend**: A new API endpoint GET /api/order/<order_id>/international-details (in blueprints/international.py) has been implemented. It fetches and consolidates customs information for line items and required compliance fields for the destination country.

- **Frontend for International Orders (New - InternationalOrderProcessor.jsx in OrderDetail.jsx)**:
    - Refactored from a simpler display component to a comprehensive processor that combines PO creation logic (similar to DomesticOrderProcessor.jsx) with international shipment details.
    - Calls the /api/order/<order_id>/international-details endpoint.
    - Displays captured and dynamically required compliance IDs.
    - Displays customs information per item.
    - Includes a section for "Order Fulfillment (Optional Drop Ship PO)" allowing selection of a supplier.
    - If a supplier is selected:
        - Displays a form to create a Purchase Order for that supplier (items, SKU, description, quantity, unit cost, PO notes).
        - Calculates and displays profitability based on entered PO costs.
        - The "Shipper" for the international label will be the selected supplier.
    - If no supplier is selected ("G1 Direct Shipment / No PO"):
        - The "Shipper" for the international label will be Global One Technology.
    - Includes input fields for final international shipment details (Weight, Dimensions, Service, Description of Goods, Branding).
    - The "Process" button now constructs a **combined payload** containing:
        - po_data: Information for generating the supplier Purchase Order (if a supplier is selected).
        - shipment_data: The complete ShipmentRequest for the UPS international API.
    - This combined payload is intended for a new backend endpoint: /api/order/<order_id>/process-international-dropship.
    - **UPS API Fixes Integrated into Frontend Payload Construction**:

        - ShipTo.Address.StateProvinceCode is now conditionally omitted for Germany (DE).
        - InternationalForms.InvoiceDate is formatted as YYYYMMDD.
        - InternationalForms.CurrencyCode is set to "USD" at the main level.
        - InternationalForms.Product[].Unit.CurrencyCode is set to "USD".
        - InternationalForms.Contacts.SoldTo.AttentionName and InternationalForms.Contacts.SoldTo.Name are populated with fallbacks, primarily using order.customer_name, and truncated to 35 characters.
        - ShipTo.AttentionName and ShipTo.Name are populated with fallbacks and truncated.
        - LabelSpecification.LabelImageFormat.Code is set to "GIF".
- **Communication & Updates**: Handles supplier email communication via Postmark and updates BigCommerce with order/shipment status for domestic orders. International updates need to be integrated into the new combined endpoint.
- **QuickBooks Integration**: Extensive QuickBooks Desktop IIF file generation for domestic orders.
- **Authentication**: Firebase Authentication with Google Sign-In and an isApproved custom claim is implemented and functional. The verify_firebase_token decorator in app.py has been updated to make user info available via flask.g without altering route function signatures.

---

3. Technology Stack & Project Layout (Updated)

(Largely as per original Handover Report, with new/modified components noted)

- **Backend (order-processing-app directory)**:
    - Language/Framework: Python 3.9+, Flask.
    - Key Libraries: SQLAlchemy, google-cloud-sql-connector, requests, ReportLab, firebase-admin, postmarker.
    - **Directory Structure**:

        - app.py: Main Flask app, shared resources (like engine, SHIPPER_EIN, COUNTRY_ISO_TO_NAME, helper functions), blueprint registration, configuration (now includes GCS_BUCKET_NAME). Modified verify_firebase_token decorator.
        - blueprints/:
            - orders.py: Manages order ingestion, display, status updates. ingest_orders_route and get_order_details updated.
            - international.py (Updated): Contains the /api/order/<order_id>/international-details GET endpoint. A new placeholder route /api/order/<order_id>/process-international-dropship (POST, OPTIONS) has been added to receive the combined PO and international shipment payload; this route requires full implementation. The /api/order/<order_id>/generate-international-shipment POST endpoint logic (for direct G1 international shipments) has been updated to expect GIF image bytes and convert them to PDF before GCS upload.
            - suppliers.py, hpe_mappings.py, quickbooks.py, reports.py, utils_routes.py (as before).
        - **Service Modules (at project root)**:
            - document_generator.py, email_service.py, iif_generator.py.
            - shipping_service.py (Updated): Contains generate_ups_international_shipment which now returns raw GIF image bytes. Contains convert_image_bytes_to_pdf_bytes. Includes diagnostic logging for payloads.
            - gcs_service.py (New): Handles file uploads to Google Cloud Storage. Contains upload_file_bytes function.
- **Frontend (order-processing-app-frontend directory)**:
    - Framework/Library: React (Vite build tool).
    - **Key Files & Structure**:
        - src/OrderDetail.jsx (Updated): Acts as a controller, conditionally rendering DomesticOrderProcessor.jsx or

InternationalOrderProcessor.jsx. Now passes suppliers prop to InternationalOrderProcessor.jsx.

        - src/components/DomesticOrderProcessor.jsx: Contains UI and logic for domestic orders.
        - src/components/InternationalOrderProcessor.jsx (Significantly Updated): Manages UI and logic for international orders, including optional PO creation for a drop-ship supplier and international shipment label generation. Constructs a combined payload. Incorporates several fixes for UPS API requirements (InvoiceDate, CurrencyCodes, AttentionNames, StateProvinceCode, LabelImageFormat).
        - src/constants/countries.js: (As per original plan, likely still needed).
- **Database**: PostgreSQL on Google Cloud SQL.
- **Cloud Platform (GCP)**: Cloud Run, Cloud SQL, Artifact Registry, Secret Manager, Cloud Storage.
- **External APIs**: BigCommerce, UPS (OAuth 2.0), FedEx (OAuth 2.0), Postmark.
- **BigCommerce Checkout Customization**:
    - custom-checkout.js: Hosted via BigCommerce Script Manager. Adds compliance ID fields to checkout.

---

4. Database Schema (Updated)

(As per "Updated" section in original Handover Report, which included compliance_info in orders, and new tables product_types, customs_info, country_compliance_fields. No further schema changes were made in this chat session).

- **orders table modifications**:
    - Added compliance_info JSONB NULL.
- **New Table 1: product_types**.
- **New Table 2: customs_info**.
- **New Table 3: country_compliance_fields** (using country_name).

---

5. Key Code Files & Their Roles (Updated)

- app.py:
    - Main Flask app, configuration (includes SHIPPER_EIN, GCS_BUCKET_NAME, COUNTRY_ISO_TO_NAME).
    - Registers international_bp.
    - verify_firebase_token decorator modified to make user info available via flask.g.
- blueprints/orders.py:
    - ingest_orders_route: Parses customer_message for compliance IDs, stores in orders.compliance_info.
    - get_order_details: Returns compliance_info.
- blueprints/international.py (Updated):
    - GET /api/order/<order_id>/international-details: Fetches consolidated international data.
    - POST /api/order/<order_id>/generate-international-shipment: (For G1 direct international shipments) Updated to handle GIF-to-PDF conversion before GCS upload.
    - POST /api/order/<order_id>/process-international-dropship (New, Placeholder): Intended to handle the combined PO data and international shipment data from InternationalOrderProcessor.jsx. **This route requires full implementation.**
- shipping_service.py (Updated):
    - generate_ups_international_shipment: Makes the UPS API call for international shipments. Now returns raw GIF image bytes and tracking number. Contains detailed payload logging.
    - convert_image_bytes_to_pdf_bytes: Utility to convert GIF (or other formats) to PDF.
    - Domestic UPS and FedEx functions as before.
- gcs_service.py (New):

    - upload_file_bytes: Uploads given bytes (e.g., PDF label) to Google Cloud Storage.
- custom-checkout.js (BigCommerce Frontend):
    - Dynamically renders compliance fields and appends data to order comments.
- OrderDetail.jsx (React Frontend - Updated):
    - Controller component, fetches order data, conditionally renders processors.
    - Now passes suppliers data as a prop to InternationalOrderProcessor.jsx.
- DomesticOrderProcessor.jsx (React Frontend):
    - UI and logic for domestic order fulfillment.
- InternationalOrderProcessor.jsx (React Frontend - Significantly Updated):
    - Calls /international-details API.
    - Displays compliance IDs and customs info.
    - Includes UI for optional "Drop Ship PO" creation (supplier selection, PO items, costs, notes).
    - Calculates profitability for drop-ship scenarios.
    - Dynamically sets Shipper address in UPS payload based on G1 or selected supplier.
    - Includes inputs for final international shipment details (weight, dimensions, service, description, branding).
    - Constructs a combined payload (po_data and shipment_data) for the new /process-international-dropship backend endpoint.
    - Correctly formats InvoiceDate, CurrencyCode(s), SoldTo.AttentionName, ShipTo.AttentionName, conditional StateProvinceCode, and requests "GIF" for LabelImageFormat.Code in the UPS payload.

---

6. Summary of Recent Development & Troubleshooting (New Section from this Chat)

This interactive session focused primarily on advancing the international shipping module, specifically enabling UPS international label generation and integrating PO creation for international drop-shipments. Key activities included:

- **Initial InternationalOrderProcessor.jsx Setup**: Began with a version of InternationalOrderProcessor.jsx focused on displaying international data and constructing a conceptual UPS payload.
- **Backend Endpoint for International Data**: The GET /api/order/<order_id>/international-details endpoint in blueprints/international.py was confirmed and used.
- **Python shipping_service.py Update**: The generate_ups_international_shipment function was created/updated to accept a detailed payload and make the UPS API call.
- **GCS Service Creation**: gcs_service.py was created to handle uploads of generated labels to Google Cloud Storage.
- **Flask App (app.py) Configuration**: Ensured GCS_BUCKET_NAME configuration and corrected the verify_firebase_token decorator to use flask.g.
- **Connecting Frontend to Backend for Label Generation**:
    - Attempted to connect InternationalOrderProcessor.jsx to a backend route (/generate-international-shipment or /process-international-dropship).
- **Extensive UPS API Troubleshooting (iterative fixes in InternationalOrderProcessor.jsx payload):**
    - **StateProvinceCode for Germany (DE)**: Resolved by conditionally omitting the field for German ShipTo addresses.
    - **InvoiceDate Format**: Corrected to YYYYMMDD format from YYYY-MM-DD.
    - **CurrencyCode**: Added "USD" to ShipmentServiceOptions.InternationalForms.CurrencyCode and to each Product[].Unit.CurrencyCode.
    - **SoldTo.AttentionName**: Logic enhanced to use order.customer_name as primary source with fallbacks, ensuring it's non-empty and truncated to 35 characters.
    - **ShipTo.AttentionName**: Logic enhanced with fallbacks to ensure it's non-empty and truncated.

    - **LabelImageFormat.Code**: Changed from "PDF" to "GIF" as UPS seemed to prefer this or reject "PDF" directly for this request type. This necessitated a backend change to convert the received GIF to PDF.
    - **Current Blocker: Missing or invalid shipper number (UPS Error 120100)**:
        - Diagnosed by logging the exact payload sent from the backend.
        - The payload was sending the literal string "UPS_BILLING_ACCOUNT_NUMBER" for Shipper.ShipperNumber and PaymentInformation.BillShipper.AccountNumber.
        - This is due to a placeholder in InternationalOrderProcessor.jsx (YOUR_G1_UPS_ACCOUNT_NUMBER) not being replaced with the actual UPS account number.
- **Integration of PO Functionality into InternationalOrderProcessor.jsx**:
    - State, useEffect hooks, handlers, and JSX for supplier selection, PO item entry (SKU, description, quantity, cost), PO notes, and profitability display were merged from DomesticOrderProcessor.jsx.
    - The handleProcessCombinedOrder function was designed to create a po_data payload and a shipment_data payload.
- **Backend Route for Combined Processing**:
    - The POST /api/order/<order_id>/process-international-dropship route was added to blueprints/international.py.
    - It's structured to receive the combined payload.
    - It currently has placeholder logic for PO creation.
    - It calls shipping_service.generate_ups_international_shipment.
    - It was updated to expect GIF bytes and call shipping_service.convert_image_bytes_to_pdf_bytes before GCS upload.
- **Import Issues Resolved**:
    - Corrected ImportError for gcs_service in international.py by ensuring gcs_service.py exists and using absolute imports.
    - Resolved TypeError for route functions by modifying the auth decorator and route function signatures to use flask.g.

---

7. Detailed Plan for Completion & Future Tasks (Updated)

This plan integrates the original handover report's phases with our current progress and identifies next steps based on this chat.

Immediate Next Steps (Fixing current issues & completing International Drop-Ship Functionality):

1. **Resolve "Missing or invalid shipper number" UPS Error:**
    - **Action (Developer)**: In InternationalOrderProcessor.jsx, at the top of the file, there is a constant:

JavaScript

```
const YOUR_G1_UPS_ACCOUNT_NUMBER =
"YOUR_ACTUAL_UPS_ACCOUNT_NUMBER_HERE";
```

Replace the placeholder string "YOUR_ACTUAL_UPS_ACCOUNT_NUMBER_HERE" with your company's actual UPS account number that is registered with UPS for API access and billing. This constant is used to populate Shipper.ShipperNumber and PaymentInformation.BillShipper.AccountNumber in the payload sent to UPS.

    - **Note**: For supplier drop-ship scenarios where the supplier uses their own UPS account, ensure the selectedSupplier.ups_account_number field in your supplier data is populated. The code currently prioritizes this if available, falling back to YOUR_G1_UPS_ACCOUNT_NUMBER.
    - **Goal**: UPS API successfully accepts the shipment request without the shipper number error.
2. **Backend: Full Implementation of POST /api/order/<order_id>/process-international-dropship in blueprints/international.py:**
    - This route receives the combinedPayload containing po_data and shipment_data.
    - **PO Processing (if po_data is present):**
        - **Action**: Implement logic to:
            - Validate po_data (supplier ID, items, costs).

        - Create records in purchase_orders and purchase_order_line_items tables. Store the generated purchase_order_id.
        - Generate the PO PDF using document_generator.generate_po_pdf() (this function will need the PO data, supplier details, and line items).
        - Save the PO PDF to Google Cloud Storage using gcs_service.upload_file_bytes().
        - (Optional, can be a later step) Email the PO PDF to the supplier using email_service.send_po_email().
    - **Goal**: Purchase order is correctly created, stored, and (optionally) sent.
- **International Shipment Processing (using shipment_data):**
    - **Action (Verify/Confirm existing logic in process_international_dropship_route):**
        - Call shipping_service.generate_ups_international_shipment(shipment_data) which returns raw_label_image_bytes (GIF) and tracking_number.
        - If raw_label_image_bytes are received, call shipping_service.convert_image_bytes_to_pdf_bytes(raw_label_image_bytes, image_format="GIF") to get final_pdf_bytes_for_gcs.
        - If final_pdf_bytes_for_gcs are available, upload them to GCS using gcs_service.upload_file_bytes(), generating a label_url.
        - Save a record to the shipments table, including order_id, carrier ('UPS'), tracking_number, label_gcs_url, service_used, and the purchase_order_id (if a PO was created for this shipment).
    - **Goal**: International shipping label is generated, stored, and shipment record created.
- **Update BigCommerce & App Order Status:**

        - **Action**: After successful PO creation (if applicable) and shipment generation:
            - Update the order status in your local G1 PO App database (e.g., to 'Processed' or 'Drop-Ship PO Created').
            - Update the order status in BigCommerce to "Shipped" (or another appropriate status) and add the tracking number using the BigCommerce API.
        - **Goal**: Order statuses are consistent across systems.
    - **Response:**
        - **Action**: Return a JSON response to the frontend including poNumber (if created), trackingNumber, and labelUrl.
        - **Goal**: Frontend is informed of the outcome.
3. **Backend: Verify GIF-to-PDF in POST /api/order/<order_id>/generate-international-shipment:**
    - **Action**: The code for this route in blueprints/international.py was also updated in the last step to include GIF-to-PDF conversion. Confirm this logic is correct and functioning for scenarios where G1 ships directly internationally without a preceding PO.
    - **Goal**: Direct G1 international shipments also store PDF labels.

Phase 3: Testing and Validation (from original Handover Report, now critical)

- **Unit & Integration Testing**:
    - **Action**: Write tests for new backend logic: /process-international-dropship endpoint, PO creation within this endpoint, interaction with shipping_service for international labels (GIF handling), and gcs_service.
    - **Goal**: Verify individual components and their integrations.
- **End-to-End (E2E) Testing**:
    - **Action**:
        - **US Domestic Order (Regression)**: Verify no changes to existing domestic fulfillment functionality (Single PO, Multi-PO, G1 Onsite).
        - **International Order - G1 Direct Shipment (No PO)**:

        - Test with a German order: Verify dynamic compliance fields, pre-filled data, customs info, and successful UPS international label (PDF)/document generation and GCS storage.
    - **International Order - Drop Ship via Supplier (PO + Label)**:
        - Test with a different international destination (e.g., Canada, to test generic compliance field if applicable).
        - Select a supplier, fill PO costs.
        - Verify PO creation in DB, PO PDF generation/storage.
        - Verify international label generation (from supplier to customer), PDF storage.
        - Verify correct Shipper details on the label.
        - Verify profitability calculation.
    - Test various scenarios of user comments + freight details + international compliance IDs.
    - Test scenarios where compliance IDs are entered at checkout, then cleared/edited in the app.
  - **Goal**: Ensure the entire international workflow is robust.
- **Regression Testing**: Thoroughly test all major existing features (domestic fulfillment, PO generation, packing slips, QuickBooks IIF files, supplier management, etc.).

Phase 4: Address Remaining Future Tasks (from original Handover Report & New Considerations)

- **FedEx "Bill RECIPIENT" Authorization**: Follow up with FedEx support.
- **Shipping Rate Estimates**: Implement UI and backend logic to fetch and display shipping rate estimates before finalizing shipment.
- **Address Validation**: Integrate an address validation service (e.g., UPS, or a third-party) for both shipper and recipient addresses to improve accuracy and reduce shipping errors.
- **Automated Testing Framework**: Build out a more comprehensive automated testing suite (e.g., Selenium, Cypress for E2E; PyTest for backend).

- **Enhanced Reporting**: Expand reporting capabilities within the app (e.g., profitability reports, shipping cost analysis).
- **Refine PO Emailing**: Fully implement and test the emailing of PO PDFs (and potentially labels/packing slips) to suppliers from the /process-international-dropship endpoint.
- **Multi-Supplier International Drop-Ship**: The current merged InternationalOrderProcessor.jsx focuses on single-supplier drop-ship for simplicity. Evaluate and implement if multi-supplier POs need to generate individual international shipments. This would significantly increase backend and UI complexity.
- **Secure Configuration Handling for Frontend**: Implement a secure way to provide sensitive configuration (like the G1 UPS Account Number) to the frontend, rather than relying on manual replacement of placeholders in JavaScript constants. This could involve a dedicated configuration API endpoint.
- **Real Supplier UPS Account Numbers**: For drop-shipping where the supplier ships on their own UPS account, ensure the suppliers table and data entry allow for storing and using these supplier-specific UPS account numbers in the Shipper.ShipperNumber and PaymentInformation.BillShipper.AccountNumber (or BillThirdParty) fields. The current frontend payload structure for ShipperDetails includes selectedSupplier.ups_account_number || YOUR_G1_UPS_ACCOUNT_NUMBER.

---

8. Key Code Assets (Updated)

The developer should have access to the project repository. Key files worked on or created during this session include:

- **InternationalOrderProcessor.jsx**: Main frontend component for handling international orders, including PO creation and international label data assembly. (Located in the Canvas with ID InternationalOrderProcessor_jsx_final_fix)
- **blueprints/international.py**: Backend Flask blueprint for international order details and shipment processing. Contains the new /process-international-dropship endpoint that needs full implementation. (File provided in the Canvas with ID international_py_gif_to_pdf_conversion)

- **shipping_service.py**: Backend service for interacting with shipping carrier APIs. Contains generate_ups_international_shipment (now expecting to return GIF bytes) and convert_image_bytes_to_pdf_bytes.
- **gcs_service.py**: New backend service for uploading files to Google Cloud Storage.
- **app.py**: Main Flask application file. verify_firebase_token decorator was updated, and GCS bucket configuration should be present.
- **OrderDetail.jsx**: Parent React component, updated to pass suppliers prop to InternationalOrderProcessor.jsx.

This report should provide a solid foundation for the next developer to understand the current state and complete the G1 PO App's international shipping capabilities. The immediate focus should be on replacing the placeholder UPS account number in the frontend, fully implementing the backend combined processing route, and then proceeding with rigorous testing.