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

Date of Report: May 27, 2025 (End of Day Update) **Compiled By:** Gemini Assistant (incorporating original report by Gemini Assistant dated May 27, 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.

The current focus has been on implementing and refining the **UPS international shipment functionality**, particularly for drop-ship scenarios.

---

2. Current State of the Application (Significantly Updated)

The application has a robust set of core features and has seen significant progress in the international shipping module. The backend, refactored with Flask Blueprints, is deployed on Google Cloud Run.

Key Achievements & Functionalities (Including Recent Developments):

- **Order Management:**
    - Robust BigCommerce order ingestion and management.
    - Parsing of international compliance IDs from order comments into the compliance_info JSONB column in the orders table.
- **Fulfillment Modes:**
    - Supports Single/Multi-Supplier POs and G1 Onsite Fulfillment for domestic orders.
    - **International Drop-Ship (PO + Label):** The backend route (/api/order/<order_id>/process-international-dropship) is now largely functional. It successfully:
        - Accepts po_data (for supplier PO) and shipment_data (for UPS international label) from the frontend.
        - **Creates Purchase Orders:** Inserts records into purchase_orders and po_line_items tables if po_data is provided. This includes storing order_id, calculated total_amount, payment_instructions (from PO notes), po_date, and created_by. PO Numbers are now generated by the application, mirroring the domestic PO strategy.
        - **Generates PO PDF:** Calls document_generator.generate_purchase_order_pdf and uploads the resulting PDF to Google Cloud Storage (GCS). The GCS path (gs://...) is stored in purchase_orders.po_pdf_gcs_path.
        - **Generates Packing Slip PDF:** Calls document_generator.generate_packing_slip_pdf for the items on the PO and uploads it to GCS. The GCS path is stored in purchase_orders.packing_slip_gcs_path (aligning with domestic supplier POs).

        - **Generates UPS International Shipping Label:** Calls shipping_service.generate_ups_international_shipment with the frontend payload. The UPS API returns label data (assumed to be PDF bytes directly, or GIF converted to PDF by shipping_service if necessary). The "Shipper" address is now correctly hardcoded to always be Global One Technology information.
        - Uploads the shipping label PDF to GCS. The HTTP URL is stored in shipments.label_gcs_url.
        - Records shipment details (tracking number, service used, GCS URLs, associated PO ID) in the shipments table.
        - **Emails Documents to Supplier:** Sends an email via email_service.send_po_email with the PO PDF, Packing Slip PDF, and Shipping Label PDF as attachments to the supplier.
        - Updates the application's order status to 'Processed'.
    - **G1 Direct International Shipments:** Basic label generation was present in /api/order/<order_id>/generate-international-shipment; however, the primary focus of recent development has been the combined drop-ship route. This direct shipment route will need review and alignment for full document/email processing.
- **Document Generation:**
    - PO PDF generation functional for domestic and now international drop-ship orders.
    - Packing Slip PDF generation functional for domestic and now international drop-ship orders.
    - Labels are generated as PDF (originally GIF from UPS for international, then converted; now assumed PDF direct or converted within shipping_service).
- **Shipping Integration (Existing Domestic):**
    - UPS: "Bill Sender" and "Bill Third Party" domestic labels functional.
    - FedEx: OAuth working; "Bill SENDER" labels tested. "Bill RECIPIENT" blocked by FedEx account config.
- **International Shipping Data Capture (Frontend):**

    - BigCommerce Checkout: Custom JavaScript (custom-checkout.js) adds fields for compliance IDs, appended to order comments.
    - Backend API (/api/order/<order_id>/international-details): Fetches customs and compliance info for the frontend.
    - Frontend (InternationalOrderProcessor.jsx):
        - Displays compliance IDs and customs info.
        - Allows supplier selection for optional drop-ship PO.
        - Calculates profitability for drop-ship POs.
        - Constructs the combined po_data and shipment_data payload for the backend, now correctly including supplierId and lineItems in po_data and ensuring the Shipper is always G1. The is_blind_drop_ship flag needs to be confirmed as part of po_data being sent.
        - Includes various UPS API fixes (Invoice Date, Currency Codes, AttentionNames, StateProvinceCode, LabelImageFormat to GIF initially, though backend now expects PDF from shipping_service).
- **Communication & Updates:**
    - Supplier email via Postmark for domestic POs. Now extended to international drop-ship POs with PO PDF, Packing Slip, and Label.
    - BigCommerce status updates for domestic orders. Needs implementation for international flow.
- **QuickBooks Integration:** IIF file generation for domestic orders. Untouched during this session.
- **Authentication:** Firebase Authentication with Google Sign-In and isApproved custom claim is functional. verify_firebase_token decorator updated to use flask.g.decoded_token.

---

3. Technology Stack & Project Layout

(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, Pillow, firebase-admin, postmarker.
    - Directory Structure:
        - app.py: Main Flask app, shared resources (DB engine, SHIPPER_EIN, COMPANY_LOGO_GCS_URI, GCS_BUCKET_NAME, helper functions), blueprint registration, config. verify_firebase_token decorator provides user info via g.decoded_token.
        - blueprints/:
            - orders.py: Manages domestic order ingestion, display, processing, status updates. PO Number generation logic.
            - international.py: Contains /api/order/<order_id>/international-details (GET) and the heavily modified /api/order/<order_id>/process-international-dropship (POST) route. Also has /api/order/<order_id>/generate-international-shipment (POST) for direct G1 international shipments (less recently touched).
            - suppliers.py, hpe_mappings.py, quickbooks.py, reports.py, utils_routes.py (as before).
        - Service Modules (at project root):
            - document_generator.py: generate_purchase_order_pdf and generate_packing_slip_pdf are key.
            - email_service.py: send_po_email is used.
            - shipping_service.py: generate_ups_international_shipment (returns PDF label bytes and tracking), convert_image_bytes_to_pdf_bytes (may be less used if UPS international returns PDF directly). FedEx functions as before.
            - gcs_service.py: upload_file_bytes function modified to remove blob.make_public() call.
            - iif_generator.py.
- **Frontend (order-processing-app-frontend directory):**
    - Framework/Library: React (Vite build tool).

    - Key Files:
        - src/OrderDetail.jsx: Controller, conditionally renders processors, passes suppliers prop.
        - src/components/DomesticOrderProcessor.jsx: UI/logic for domestic orders.
        - src/components/InternationalOrderProcessor.jsx: UI/logic for international orders. Collects data for PO and shipment. Constructs combined payload. Shipper address fixed to always be G1. is_blind_drop_ship flag needs to be confirmed in its po_data payload.
- **Database:** PostgreSQL on Google Cloud SQL.
- **Cloud Platform (GCP):** Cloud Run, Cloud SQL, Artifact Registry, Secret Manager, Cloud Storage (bucket: g1-po-app-documents).
- **External APIs:** BigCommerce, UPS (OAuth 2.0), FedEx (OAuth 2.0), Postmark.
- **BigCommerce Checkout Customization:** custom-checkout.js for compliance ID fields.

---

4. Database Schema (Updated with recent changes)

(Original schema details for product_types, customs_info, country_compliance_fields as per prior report).

- **orders Table:**
    - compliance_info (JSONB): Stores compliance IDs from checkout.
    - Other fields as per original schema (e.g., sales_rep_email, shipping/billing details).
- **purchase_orders Table:**
    - id (SERIAL PRIMARY KEY)
    - po_number (TEXT - Application Generated)
    - order_id (INTEGER, FK to orders.id)
    - supplier_id (INTEGER, FK to suppliers.id)
    - po_date (TIMESTAMP WITH TIME ZONE)

  - payment_instructions (TEXT - stores PO notes)
  - status (TEXT - e.g., "PO Sent")
  - total_amount (NUMERIC(10,2))
  - created_by (TEXT - stores user's email)
  - created_at (TIMESTAMP WITH TIME ZONE)
  - updated_at (TIMESTAMP WITH TIME ZONE)
  - po_pdf_gcs_path (TEXT - stores gs:// URI for PO PDF)
  - packing_slip_gcs_path (TEXT - stores gs:// URI for Packing Slip PDF associated with this PO)
- **po_line_items Table (Formerly purchase_order_line_items - name corrected):**
  - id (SERIAL PRIMARY KEY)
  - purchase_order_id (INTEGER NOT NULL, FK to purchase_orders.id ON DELETE CASCADE)
  - sku (TEXT - corrected from product_sku)
  - description (TEXT)
  - quantity (INTEGER NOT NULL)
  - unit_cost (NUMERIC(10, 2) NOT NULL)
  - *(Consider adding condition, created_at, updated_at, original_order_line_item_id if needed, as seen in user's screenshot of an existing po_line_items table, though current inserts don't use them).*
- **shipments Table:**
  - id (SERIAL PRIMARY KEY)
  - order_id (INTEGER, FK to orders.id)
  - purchase_order_id (INTEGER, FK to purchase_orders.id, NULLABLE)
  - tracking_number (TEXT, has UNIQUE constraint shipments_tracking_number_key)
  - carrier (TEXT)

    - label_gcs_url (TEXT - stores HTTP URL for shipping label PDF)
    - service_used (VARCHAR(10) - stores UPS/FedEx service code)
    - packing_slip_gcs_path (TEXT - stores gs:// URI for packing slip, primarily for G1 direct shipments or if a shipment-specific packing slip is ever needed distinct from the PO's packing slip).
    - created_at (TIMESTAMP WITH TIME ZONE)
    - *(Other columns from user's screenshot like method, weight_lbs, label_pdf_storage_path, packing_slip_pdf_storage_path, shipped_at, updated_at, shipping_method_name, label_gcs_path exist but are not all actively managed by the international drop-ship route in its current form. Their usage should be reviewed for consistency across domestic and international flows.)*
- **suppliers Table:**
    - Ensure it has email, name, address_line1, city, state, zip, country (should store 2-letter ISO code), payment_terms, defaultponotes for PDF generation and email.
- **users Table:**
    - **Not created.** The decision was made *not* to use sales_rep_name on the PO, so the subquery to the users table was removed from international.py. If user/sales rep details are needed elsewhere or in the future, this table might need to be created.

---

5. Key Code Files & Their Roles (Updated with recent changes)

- **app.py:**
    - Main Flask app, configuration (including SHIPPER_EIN, COMPANY_LOGO_GCS_URI, GCS_BUCKET_NAME).
    - Registers international_bp.
    - verify_firebase_token decorator updated to use g.decoded_token.
- **blueprints/international.py:**

  - get_international_details: Fetches consolidated international data for frontend display. SQL query modified to remove users table dependency.
  - process_international_dropship_route:
    - Orchestrates the international drop-ship workflow.
    - Handles combined po_data and shipment_data.
    - Generates PO numbers application-side (max numeric + 1 logic).
    - Inserts into purchase_orders (with order_id, total_amount, payment_instructions, po_date, created_by, po_number) and po_line_items.
    - Calls document_generator.generate_purchase_order_pdf and document_generator.generate_packing_slip_pdf.
    - Uploads PO and Packing Slip PDFs to GCS, storing gs:// paths in purchase_orders.po_pdf_gcs_path and purchase_orders.packing_slip_gcs_path respectively, aligning with domestic PO GCS structure.
    - Calls shipping_service.generate_ups_international_shipment (expects PDF label bytes).
    - Uploads shipping label PDF to GCS, stores HTTP URL in shipments.label_gcs_url.
    - Records shipment in shipments table (associating purchase_order_id, and setting its packing_slip_gcs_path to None for PO-related shipments as the PS is linked to the PO).
    - Calls email_service.send_po_email with PO PDF, shipping label PDF, and packing slip PDF as attachments.
  - generate_international_shipment_route: For G1 direct international shipments. This route's document generation and emailing capabilities are less developed compared to the drop-ship route and need review.
- **document_generator.py:**
  - generate_purchase_order_pdf: Used to create PO PDFs. Expects order_data, supplier_data, po_number, po_date, po_items (with unit_cost), payment_terms, payment_instructions, logo_gcs_uri.

  - generate_packing_slip_pdf: Used to create Packing Slip PDFs. Expects order_data, items_in_this_shipment (with name, quantity, sku), is_blind_slip, logo_gcs_uri, etc.
- **email_service.py:**
  - send_po_email: Sends email to supplier with attachments. Attachments are prepared with Base64 encoded content.
- **shipping_service.py:**
  - generate_ups_international_shipment: Makes the UPS API call. Expects the frontend shipment_data payload. Returns PDF label bytes and tracking number. Shipper is always G1.
- **gcs_service.py:**
  - upload_file_bytes: Modified to remove blob.make_public() to work with Uniform Bucket Level Access. Returns the public HTTP URL.
- **src/components/InternationalOrderProcessor.jsx (React Frontend):**
  - Collects supplier selection, PO item costs, PO notes for po_data.
  - Collects weight, dimensions, service, description, branding for shipment_data.
  - Shipper for UPS payload is now hardcoded to G1 Technology details.
  - Sends is_blind_drop_ship boolean in po_data (developer needs to confirm this is implemented).
- **src/OrderDetail.jsx (React Frontend):**
  - Passes suppliers prop to InternationalOrderProcessor.

---

6. Summary of Recent Development & Troubleshooting (This Chat Session)

This session focused on getting the UPS international drop-ship functionality fully operational, from backend processing to document generation and emailing.

- **Initial State:** UPS international label generation was the target. An error log was provided showing GCS upload and DB write failures.

- **GCS Upload Fix:** Resolved google.api_core.exceptions.BadRequest: 400 ... Cannot get legacy ACL for an object when uniform bucket-level access is enabled. by removing blob.make_public() from gcs_service.py.
- **Database Schema Corrections (Iterative):**
  - Added label_gcs_url to shipments table.
  - Added service_used to shipments table.
  - Addressed shipments_tracking_number_key duplicate key error by advising deletion of test data.
  - Added created_by to purchase_orders table.
  - Corrected purchase_order_line_items table name to po_line_items and product_sku column to sku in international.py INSERT statement to match user's existing schema.
  - Addressed relation "users" does not exist by removing the sales_rep_name subquery from SQL in get_international_details and process_international_dropship_route in international.py as per user's decision not to include it on the PO.
- **PO Number Generation:**
  - Initially, process_international_dropship_route relied on RETURNING po_number. This led to a ValueError: Failed to create PO or retrieve PO ID/Number from database. (and earlier "PO None" symptoms in logs/emails).
  - Fixed by implementing application-side PO number generation in international.py, mirroring the strategy in orders.py (querying max existing numeric PO, starting at 200001).
  - Corrected the max_po_query to use CAST(po_number AS TEXT) ~ '^[0-9]+$' within a subquery to resolveoperator does not exist: integer ~ unknown` .
- **purchase_orders Table Data:** Modified international.py to insert order_id, po_date, total_amount, and use po_notes for payment_instructions column, aligning with orders.py.
- **Frontend Payload (InternationalOrderProcessor.jsx):**
  - Addressed ValueError: Supplier ID and line items are required for PO. by guiding the user to add state management and correct po_data construction

    (for supplierId, lineItems, poNotes) in InternationalOrderProcessor.jsx, using DomesticOrderProcessor.jsx as a model.

- **Business Logic Correction (Shipper Address):** Updated InternationalOrderProcessor.jsx to ensure the Shipper in the UPS payload is *always* Global One Technology, not the supplier, even for drop-ships. This also resolved the Invalid OriginCountry UPS API error (by using G1's "US" country code).
- **Logo on PDFs:** Resolved PIL.UnidentifiedImageError for SVG logo by advising conversion of the logo to PNG format and updating the COMPANY_LOGO_GCS_URI environment variable.
- **Document Generation Integration in international.py:**
    - Implemented PO PDF generation using document_generator.generate_purchase_order_pdf.
    - Implemented Packing Slip PDF generation using document_generator.generate_packing_slip_pdf.
    - Logic added to upload both to GCS.
- **GCS Path Alignment:** Updated international.py to save PO PDF and Packing Slip PDF to GCS using a directory structure similar to orders.py (processed_orders/order_{BC_ORDER_ID}_PO_{PO_NUMBER}{BLIND_SUFFIX}/).
    - PO PDF path stored in purchase_orders.po_pdf_gcs_path.
    - Packing Slip PDF path (for POs) stored in purchase_orders.packing_slip_gcs_path.
- **Email Service Integration:**
    - Integrated email_service.send_po_email to send an email to the supplier with the PO PDF, Shipping Label PDF, and Packing Slip PDF as attachments.
    - Corrected variable usage to ensure the actual generated_po_number is used in logs, filenames, and email content.

The process_international_dropship_route in international.py is now quite comprehensive. The last provided log showed a 200 OK status, indicating successful execution of this route.

---

7. Detailed Plan for Completion & Future Tasks

- **A. Immediate Next Steps (Verification & Minor Code Confirmations):**

    1. **Verify po_number in Database for Last Test:** Although the last test was 200 OK, double-check the purchase_orders table for the PO created during that test. Confirm that the po_number column was correctly populated by the application-generated number and is not None or empty. This ensures the "PO None" issue is fully resolved.

    2. **Confirm is_blind_drop_ship in Frontend Payload:** In InternationalOrderProcessor.jsx, ensure the isBlindDropShip state variable is correctly included in the po_data object when the request is sent to /api/order/<order_id>/process-international-dropship. The backend (international.py) now relies on po_data.get('is_blind_drop_ship', False).

    3. **Verify Email Attachment Encoding:** The international.py currently uses base64.b64encode(pdf_bytes).decode('utf-8') for email attachment Content. Review email_service.py (and Postmark documentation) to confirm if it expects Base64 encoded strings or raw bytes. The email_service.py provided seems to pass Content directly, implying the Postmark library handles encoding. If so, remove the Base64 encoding step in international.py for email attachments.

    4. **Frontend Display of PDF Links:** The backend now returns poPdfUrl and packingSlipPdfUrl (these are HTTP URLs). The InternationalOrderProcessor.jsx should be updated to display these links to the user upon successful processing.

- **B. BigCommerce Integration (Backend Task):**

    - As per the original plan, after successful PO creation and shipment generation in process_international_dropship_route:

        - Implement API calls to BigCommerce to update the order status (e.g., to "Shipped" or a custom status).

        - Add the UPS tracking number to the order in BigCommerce.

        - Refer to shipping_service.py which contains create_bigcommerce_shipment and set_bigcommerce_order_status functions that can be leveraged.

- **C. Thorough End-to-End Testing (Phase 3 from original report):**

    1. **International Drop-Ship (PO + Label - Current Focus):**

        - Test with multiple international destinations.
        - Verify all document contents (PO, Packing Slip, Shipping Label/Commercial Invoice) for accuracy (addresses, items, PO numbers, branding for blind vs. non-blind).
        - Confirm GCS storage paths and filenames match the new processed_orders/... structure.
        - Confirm database records are complete (all fields in purchase_orders, po_line_items, shipments including GCS paths).
        - Verify email to supplier has all three correct PDF attachments.
        - Test UI profitability calculations.
        - Test edge cases with compliance IDs (entered at checkout, edited in app, exempted).
    2. **International G1 Direct Shipment (No PO):**
        - The /api/order/<order_id>/generate-international-shipment route is the current entry point for this.
        - **Action:** This route needs to be reviewed and potentially enhanced to match the process_international_dropship_route in terms of complete document generation (Packing Slip, potentially a simplified "internal" PO or just a record of shipment), GCS storage (using consistent naming), and internal/sales notifications (email).
        - Test with a German order: verify dynamic compliance fields, label generation, GCS storage.
    3. **Domestic Order Processing (Regression Testing):**
        - Thoroughly test all existing domestic fulfillment flows (Single Supplier PO, Multi-Supplier PO, G1 Onsite Fulfillment) to ensure no regressions were introduced. Verify POs, Packing Slips, Labels, QuickBooks IIF files, and supplier emails for domestic orders.
- **D. Implement UPS Shipment Void Functionality (Critical for Production):**
    - **Action:** Add a new function void_ups_shipment(tracking_number) to shipping_service.py.

    - This function needs to call the UPS Void Shipment API endpoint (e.g., /{version}/shipments/void/{trackingNumber} or similar for the correct API version).
    - Create a new backend route (e.g., in utils_routes.py or shipping_service.py if it had its own blueprint) that can be called from the UI (perhaps with admin privileges) to void a shipment by its tracking number.
    - This is essential for correcting errors or voiding test shipments made with production credentials.
- **E. 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.
    - **Address Validation Service Integration:** For shipper and recipient addresses.
    - **Automated Testing Framework:** Build out PyTest for backend, Selenium/Cypress for E2E.
    - **Enhanced Reporting:** Profitability, shipping costs, etc..
    - **Multi-Supplier International Drop-Ship:** Evaluate and implement if needed (significant complexity increase).
    - **Secure Configuration Handling for Frontend:** If any sensitive data ever needs to be passed to the frontend beyond what's already handled by backend-for-frontend patterns.

---

8. Key Code Assets

The developer should have access to the project repository. The most recently modified and crucial files during this session were:

- **blueprints/international.py**: Contains the primary backend logic for international order processing. The process_international_dropship_route function is now very comprehensive.
- **InternationalOrderProcessor.jsx (Frontend Component)**: Responsible for UI and payload construction for international orders.

- **document_generator.py**: Contains generate_purchase_order_pdf and generate_packing_slip_pdf.
- **email_service.py**: Contains send_po_email.
- **shipping_service.py**: Contains generate_ups_international_shipment.
- **gcs_service.py**: Modified for GCS uploads.
- **app.py**: Main application, configuration.
- **Database Schema**: As detailed in section 4.

---

9. Immediate Next Steps for the New Developer

1. **Verify PO Numbering & Email Attachments:**
   - Confirm the po_number value in the purchase_orders table for the last successful test (Order 253, around May 27, 2025, 16:11 - 16:43 server time from logs). Ensure it's not None.
   - Confirm the filenames and content of the PDFs in GCS from that test.
   - Confirm the received email subject and all three attachments (PO, Label, Packing Slip) are correct.
   - Verify email_service.py's attachment Content requirement (raw bytes vs. Base64) and adjust international.py if the current Base64 encoding is incorrect for Postmark.
2. **Frontend - is_blind_drop_ship Payload:**
   - Ensure InternationalOrderProcessor.jsx correctly includes the is_blind_drop_ship: boolean field within the po_data object when sending the request to the backend.
3. **Frontend - Display PDF Links:**
   - Update InternationalOrderProcessor.jsx to display the poPdfUrl and packingSlipPdfUrl returned by the backend after successful processing.
4. **Implement BigCommerce Updates:**

    - Add the logic to international.py (within process_international_dropship_route) to update the order status and add tracking information in BigCommerce after successful processing.

5. **Commence Thorough End-to-End Testing:** Follow the detailed testing plan outlined in section 7.C.
6. **Implement UPS Void Functionality:** Add the void_ups_shipment function to shipping_service.py and a corresponding backend route. This is crucial before extensive production API use.

This report should provide a solid foundation for the next developer to understand the current state, the progress made, and the remaining tasks to complete the G1 PO App's international shipping capabilities and overall project goals. The immediate focus should be on the verifications and BigCommerce integration, followed by comprehensive testing and implementing the void functionality.

Sources