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

Date of Report: May 27, 2025

Compiled By: Gemini Assistant (incorporating original report by Gemini Assistant dated May 24, 2025)

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 is now being built out.
- **Document Generation**: PDF generation for Purchase Orders and Packing Slips, including a "blind drop ship" version.
- **Shipping Integration (Existing)**:
    - 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 using a ||| separator.
    - **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 (description, harmonized code, country of origin) for each line item by looking up SKU -> hpe_part_mappings -> option_pn -> product_types -> customs_info.
        - Required compliance fields for the destination country from the country_compliance_fields table.

        - The static Shipper's EIN.
- **Frontend for International Orders (New - In Progress)**:
    - OrderDetail.jsx has been refactored to act as a controller, conditionally rendering DomesticOrderProcessor.jsx or InternationalOrderProcessor.jsx.
    - InternationalOrderProcessor.jsx currently:
        - Calls the new /api/order/<order_id>/international-details endpoint.
        - Displays the destination country.
        - Displays compliance IDs captured from checkout (from order.compliance_info).
        - Dynamically renders a form section for "Required Tax/Compliance IDs" based on data from country_compliance_fields, pre-filling values from checkout or the Shipper's EIN.
        - Displays a table of "Customs Information per Item."
        - Includes input fields for "Shipment Details" (Weight, Dimensions, Service).
        - Has a button to "Prepare International Shipment Data (Log to Console)" which assembles a conceptual payload for the UPS API.
- **Communication & Updates**: Handles supplier email communication via Postmark and updates BigCommerce with order/shipment status for domestic orders.
- **QuickBooks Integration**: Extensive QuickBooks Desktop IIF file generation.
- **Authentication**: Firebase Authentication with Google Sign-In and an isApproved custom claim is implemented and functional.

---

3. Technology Stack & Project Layout (Updated)

(Largely as per original Handover Report, with new frontend components)

- **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, blueprint registration, configuration (including new SHIPPER_EIN and COUNTRY_ISO_TO_NAME dictionary).
        - blueprints/:
            - orders.py: Manages order ingestion, display, and status updates. ingest_orders_route and get_order_details have been updated.
            - international.py (New): Contains the /api/order/<order_id>/international-details endpoint.
            - suppliers.py, hpe_mappings.py, quickbooks.py, reports.py, utils_routes.py (as before).
        - Service Modules: document_generator.py, email_service.py, iif_generator.py, shipping_service.py (will require significant updates for international UPS).
- **Frontend (order-processing-app-frontend directory)**:
    - Framework/Library: React (Vite build tool).
    - Key Files & Structure:
        - src/OrderDetail.jsx: Refactored to a "controller" component.
        - src/components/DomesticOrderProcessor.jsx (New): Contains existing UI logic for processing domestic orders.
        - src/components/InternationalOrderProcessor.jsx (New): Contains UI and logic for processing international orders, currently displaying fetched data and input forms.
        - src/constants/countries.js (New, as per plan): To be populated for UI dropdowns if 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 (URL type). Contains logic to dynamically add compliance ID fields to the checkout and append data to order comments.

---

4. Database Schema (Updated)

Includes existing tables plus the new tables for international shipping and a new column in the orders table.

- **Existing orders table modifications**:
  - Added a new column: compliance_info JSONB NULL (to store the JSON object of captured compliance IDs).
- **New Table 1: product_types**
  - Purpose: Maps option_pn from hpe_part_mappings to a human-readable product_type.
  - SQL:

```
SQL

CREATE TABLE product_types (

  id SERIAL PRIMARY KEY,

  option_pn VARCHAR(255) UNIQUE NOT NULL,

  product_type VARCHAR(255) NOT NULL

);

COMMENT ON TABLE product_types IS 'Maps an option_pn to a specific product_type string.';

CREATE INDEX idx_option_pn ON product_types(option_pn);
```

- **New Table 2: customs_info**
  - Purpose: Stores customs information for a given product_type.
  - SQL:

```
SQL
```

```sql
CREATE TABLE customs_info (
  id SERIAL PRIMARY KEY,
  product_type VARCHAR(255) UNIQUE NOT NULL,
  customs_description TEXT NOT NULL,
  harmonized_tariff_code VARCHAR(255) NOT NULL,
  default_country_of_origin VARCHAR(2) NOT NULL DEFAULT 'US'
);
COMMENT ON TABLE customs_info IS 'Stores customs information for a given product_type string.';
CREATE INDEX idx_product_type_customs ON customs_info(product_type);
```

- **New Table 3: country_compliance_fields**
  - Purpose: Defines dynamic, country-specific compliance ID requirements. The column queried is country_name (not country_iso2 as originally planned in the PDF, due to user's database structure).
  - SQL (Reflecting country_name based on current implementation):

SQL

```sql
CREATE TABLE country_compliance_fields (
  id SERIAL PRIMARY KEY,
  country_name VARCHAR(255) NOT NULL, -- Changed from country_iso2
  field_label VARCHAR(255) NOT NULL,
  id_owner VARCHAR(50) NOT NULL,
  is_required BOOLEAN NOT NULL DEFAULT true,
  has_exempt_option BOOLEAN NOT NULL DEFAULT false
);
COMMENT ON TABLE country_compliance_fields IS 'Defines required compliance ID fields for international shipments based on destination country name.';
```

CREATE INDEX idx_country_name_compliance ON
country_compliance_fields(country_name); -- Index on country_name

- *Note to Developer:* The SQL above for country_compliance_fields uses country_name. The originally provided PDF had country_iso2. The current Python code in international.py queries this table using the full country name after converting the ISO code. Ensure data population and any future queries are consistent with using country_name.

---

5. Key Code Files & Their Roles (Updated)

- **app.py**: Main Flask app, configuration (now includes SHIPPER_EIN = "421713620" and COUNTRY_ISO_TO_NAME dictionary), blueprint registration (including international_bp), shared helper functions (get_hpe_mapping_with_fallback, get_country_name_from_iso, convert_row_to_dict, make_json_safe).
- **blueprints/orders.py**:
  - ingest_orders_route: Updated to parse customer_message for ||| separated compliance ID blocks, store them in the orders.compliance_info JSONB column, and correctly separate user notes and freight details.
  - get_order_details: Updated to select and return the compliance_info field (parsed as a dictionary) to the frontend.
- **blueprints/international.py (New)**:
  - Contains the GET /api/order/<order_id>/international-details endpoint. This endpoint implements the data lookup workflow:
    1. Fetches order's customer_shipping_country_iso2.
    2. Converts ISO to full country name.
    3. Queries country_compliance_fields using the full country name and * wildcard.
    4. For each line item: SKU -> hpe_part_mappings.option_pn -> product_types.product_type -> customs_info details.
    5. Returns consolidated JSON: line_items_customs_info, required_compliance_fields, shipper_ein.
- **shipping_service.py**:

  - Currently handles domestic UPS and FedEx label generation.
  - **Requires significant updates** to create a new function (e.g., generate_ups_international_shipment) to accept detailed international data and build the complex UPS API request payload, including InternationalForms for customs.
- **custom-checkout.js (BigCommerce Frontend)**:
  - Loaded via BigCommerce Script Manager (URL type).
  - Contains complianceFieldsConfig to define checkout fields per country (including a * wildcard for a generic "Tax ID / Registration Number").
  - Dynamically renders these fields on the BigCommerce checkout page.
  - Appends captured compliance IDs to the order comments, formatted as ... ||| [Label1: Value1; Label2: Value2;];.
- **OrderDetail.jsx (React Frontend)**:
  - Refactored to a controller component. Fetches order data (including compliance_info).
  - Conditionally renders DomesticOrderProcessor.jsx or InternationalOrderProcessor.jsx based on order.is_international.
  - Displays profit for *already processed* orders.
- **DomesticOrderProcessor.jsx (React Frontend - New)**:
  - Contains the UI and logic for domestic order fulfillment (Single PO, Multi-PO, G1 Onsite) moved from the old OrderDetail.jsx.
  - Manages its own state for PO items, costs, and calculates/displays profit for *pending* domestic orders.
- **InternationalOrderProcessor.jsx (React Frontend - New)**:
  - Calls /api/order/<order_id>/international-details to get data.
  - Displays captured compliance IDs (from order.compliance_info).
  - Renders a dynamic form for "Required Tax/Compliance IDs" based on internationalApiDetails.required_compliance_fields, pre-filling values.

    - Displays a table of "Customs Information per Item" from internationalApiDetails.line_items_customs_info.
    - Includes input fields for "Shipment Details" (weight, dimensions, service).
    - The handleProcessInternationalOrder function currently assembles and logs a conceptual payload for the UPS API.

---

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

This interactive session focused on implementing the international shipping module:

- **BigCommerce Checkout Script (custom-checkout.js)**:
    - Developed JavaScript to dynamically add compliance ID fields to the BigCommerce checkout page based on destination country.
    - Implemented logic to append these IDs to the order comments field using ||| as a separator and a [Label: Value; ...]; format.
    - Troubleshooting included:
        - Content Security Policy (CSP) issues with inline scripts. Resolved by advising the use of BigCommerce Script Manager with "URL" type, and then guiding the user to upload the custom-checkout.js file via WebDAV when the theme editor was found to be read-only for JS files.
        - Subresource Integrity (SRI) hash requirement for scripts loaded via Script Manager. A hash was generated and provided.
        - Correcting the custom-checkout.js to ensure focus wasn't lost on input fields and that cleared fields were correctly updated in order comments.
- **Backend Python (orders.py)**:
    - Refined ingest_orders_route multiple times to correctly parse the customer_message:
        - Isolate the compliance ID block (e.g., ||| [IDs];).
        - Separate actual user notes from freight details (separated by ||).
        - Store parsed compliance IDs into a new compliance_info JSONB column in the orders table.

- **Backend Python (international.py)**:
    - Created the GET /api/order/<order_id>/international-details endpoint.
    - Implemented the multi-step lookup: SKU -> hpe_part_mappings.option_pn -> product_types.product_type -> customs_info details.
    - Implemented lookup of country_compliance_fields based on destination country (using full country name) and * wildcard.
    - Troubleshooting included:
        - Initial 404 errors (fixed by ensuring blueprint was registered and Flask restarted).
        - Firebase token authentication errors (user was initially using an API key).
        - Database error column "country_iso2" does not exist in country_compliance_fields. Resolved by confirming the user's table uses country_name and updating the Python query to match.
        - Case-sensitivity issue for product_type lookup between product_types and customs_info tables (resolved by user data correction).
- **Backend Python (app.py)**:
    - SHIPPER_EIN = "421713620" added as a configuration constant.
    - COUNTRY_ISO_TO_NAME dictionary confirmed/added.
    - get_country_name_from_iso and get_hpe_mapping_with_fallback helper functions confirmed.
- **Frontend React (OrderDetail.jsx, InternationalOrderProcessor.jsx, DomesticOrderProcessor.jsx)**:
    - Began refactor of OrderDetail.jsx into a controller.
    - Created InternationalOrderProcessor.jsx.
        - It successfully calls the new international details API endpoint.
        - It displays captured compliance IDs, dynamically required compliance fields (with pre-filling), customs information per item, and placeholders for shipment details.

- The plan for DomesticOrderProcessor.jsx (to house existing domestic logic) is set.
- Discussed and planned how the "Profitability Analysis" feature will work across the refactored components.

---

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

This plan integrates the original handover report's phases with our current progress.

Immediate Next Steps (Completing International Shipping Module - Phase 2 & 5.2 from Handover Report):

1. **Frontend: Complete InternationalOrderProcessor.jsx Data Input & Payload Assembly:**
   - **Action**:
     - Ensure all necessary input fields are present and state-managed:
       - Package Dimensions (Length, Width, Height) - Add state and inputs.
       - Finalize selection for International Shipping Service (map to UPS service codes).
       - Add UI for Payment Information (Bill Shipper is default; consider UI for Bill Receiver/Third Party if needed, capturing account numbers and postal codes).
     - Fully implement the handleProcessInternationalOrder function:
       - Gather all data from orderData, internationalApiDetails, dynamicComplianceValues, and new input field states.
       - Construct the complete, accurate JSON payload for the UPS international shipment API, using the example provided by the user and UPS documentation as guides. This includes the InternationalForms object with product details for the Commercial Invoice.
       - For now, continue to console.log this fully assembled payload for verification.

   - **Goal**: InternationalOrderProcessor.jsx can collect all data and build a UPS-ready JSON payload.

2. **Backend: Create API Endpoint for International Shipment Processing & Update shipping_service.py**:
   - **Action (New API Route)**: Create a new backend API endpoint (e.g., POST /api/order/<order_id>/generate-international-shipment in international.py or a new shipping_blueprint.py). This endpoint will receive the JSON payload from InternationalOrderProcessor.jsx.
   - **Action (shipping_service.py Update)**:
     - Create a new function (e.g., generate_ups_international_shipment) in shipping_service.py.
     - This function will accept the detailed payload from the new API route.
     - It must build the precise API request for the UPS Shipping API, including authentication, InternationalForms (Commercial Invoice data based on line items, customs details, compliance IDs, etc.).
     - It will make the call to the UPS API.
     - It will process the UPS response, extracting the tracking number, label image data (e.g., Base64 encoded GIF/PDF), and any links to generated customs documents.
     - This function should then save label/document PDFs to Google Cloud Storage and store their GCS URIs in the shipments (and potentially purchase_orders for PO-related documents if applicable) table.
     - Return the tracking number and GCS URIs (or signed URLs) for the documents to the calling API route.
   - **Action (Backend Route)**: The new API route, after calling shipping_service.py, will return the tracking number and document URLs to the frontend.
   - **Goal**: Backend can receive shipment data, call UPS API for international shipment, save artifacts, and return results.

3. **Frontend: Connect InternationalOrderProcessor.jsx to New Backend Endpoint & Handle Response**:

   - **Action**:
      - Modify handleProcessInternationalOrder in InternationalOrderProcessor.jsx to POST the assembled payload to the new backend shipment endpoint.
      - On successful response from the backend:
         - Display the tracking number.
         - Provide links for the user to download/view the shipping label and commercial invoice.
         - Call WorkspaceOrderAndSuppliers(null, true) to refresh order data (e.g., to show new shipment record, updated status).
         - Set processSuccess state to true and display a success message.
      - Handle any errors returned from the backend and display them using setProcessError.
   - **Goal**: User can click a button in InternationalOrderProcessor.jsx, generate an international UPS shipment, and get back the label/tracking.

4. **Update BigCommerce & App Order Status**:
   - **Action (Backend)**: After a successful international shipment generation in shipping_service.py (or the calling route), update the order status in your local G1 PO App database (e.g., to 'Processed' or 'Completed Offline').
   - **Action (Backend)**: Implement logic to update the order status in BigCommerce to "Shipped" (or another appropriate status) and add the tracking number using the BigCommerce API (similar to existing domestic shipment updates).
   - **Goal**: Order statuses are consistent across systems.

Phase 3: Testing and Validation (from Handover Report)

- **Unit & Integration Testing**: Write tests for new backend logic (API endpoint, shipping_service.py international functions, data parsing).
- **End-to-End (E2E) Testing**:
   - **US Domestic Order (Regression)**: Verify no changes to existing functionality.

    - **German Order (Full Feature)**: Verify dynamic fields, pre-filled data, customs info, and successful UPS international label/document generation.
    - **Other International Order (e.g., Canada using generic field)**: Verify UI, generic compliance field, and successful UPS label/document generation.
    - Test all scenarios of user comments + freight details + compliance IDs.
    - Test scenarios where compliance IDs are entered, then cleared/edited.
- **Regression Testing**: Test all major existing features (domestic fulfillment, PO generation, QuickBooks IIF files).

Phase 4: Address Remaining Future Tasks (from Handover Report)

- **FedEx "Bill RECIPIENT" Authorization**: Follow up with FedEx support.
- **Shipping Rate Estimates**: Implement.
- **Address Validation**: Integrate.
- **Automated Testing Framework**: Build out.
- **Enhanced Reporting**: Expand.

---

8. Key Code Assets (Updated)

(The developer should refer to the latest versions of these files as developed during this chat session. The developer can refer to the project repository and the original Handover Report.)

- **custom-checkout.js** (for BigCommerce Script Manager)
- **app.py** (specifically new config values and helper functions)
- **blueprints/orders.py** (specifically the ingest_orders_route and get_order_details functions)
- **blueprints/international.py** (full file for the new API endpoint)
- **shipping_service.py** (will need significant additions for international UPS payload and API call)
- **src/OrderDetail.jsx** (React controller component)
- **src/components/DomesticOrderProcessor.jsx** (React component)

- **src/components/InternationalOrderProcessor.jsx** (React component)