Project Handover Report: G1 PO App

- **Date of Report:** May 24, 2025
- **Compiled By:** Gemini Assistant
- **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

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

Key Achievements & Functionalities:

- **Order Management:** Robust BigCommerce order ingestion and management.

- **Fulfillment Modes:** Supports Single/Multi-Supplier POs and G1 Onsite Fulfillment.
- **Document Generation:** PDF generation for Purchase Orders and Packing Slips, including a "blind drop ship" version with a changed font ("Eloquia").
- **Shipping Integration:**
    - **UPS:** "Bill Sender" label generation is functional. The "Bill Third Party" functionality is now also working correctly after a fix to the PaymentInformation block in shipping_service.py was implemented.
    - **FedEx:** OAuth is working and "Bill SENDER" labels are tested. "Bill RECIPIENT" functionality is implemented but blocked by a FedEx API error ("SHIPMENT.ACCOUNTNUMBER.UNAUTHORIZED") that requires configuration changes on the FedEx account itself.
- **Communication & Updates:** Handles supplier email communication via Postmark and updates BigCommerce with order/shipment status.
- **QuickBooks Integration:** Extensive QuickBooks Desktop IIF file generation.
- **Authentication:** Firebase Authentication with Google Sign-In and an isApproved custom claim is implemented.
- **Frontend (OrderDetail.jsx):** The UI was recently updated to rename and relocate the branding/shipping dropdown. A critical bug causing a blank page (currentSelectedShippingOption is not defined) has been resolved.

3. Technology Stack & Project Layout

This reflects the current architecture, including planned changes.

- **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.
        - blueprints/: Contains modules for orders, suppliers, quickbooks, etc.
        - Service Modules: document_generator.py, email_service.py, iif_generator.py, shipping_service.py.

- **Frontend (order-processing-app-frontend directory):**
    - **Framework/Library:** React (Vite build tool)
    - **Key Files & Planned Structure:**
        - src/OrderDetail.jsx: To be refactored into a "controller" component that fetches data and conditionally renders a processor component.
        - **src/components/DomesticOrderProcessor.jsx (New):** Will contain the existing UI logic for processing domestic (US) orders.
        - **src/components/InternationalOrderProcessor.jsx (New):** Will contain all new UI and logic for processing international orders.
        - **src/constants/countries.js (New):** A constants file exporting an array of all countries and their 2-letter ISO codes to populate UI dropdowns.
- **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.

4. Database Schema

This includes the existing orders table and the **three new tables** required for the international shipping feature.

- **New Table 1: product_types**
    - **Purpose:** Maps an option_pn from the hpe_part_mappings table 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 the detailed customs information for a given product_type.
    - **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 for international shipments.
    - **SQL:**

SQL

CREATE TABLE country_compliance_fields (

id SERIAL PRIMARY KEY,

country_iso2 VARCHAR(2) NOT NULL,

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.';
CREATE INDEX idx_country_iso2 ON country_compliance_fields(country_iso2);
```

5. Key Code Files & Their Roles

This section describes the roles of key files including the planned refactoring.

- **blueprints/orders.py:** The process_order_route will need to be updated to handle the new international data fields passed from the frontend.
- **shipping_service.py:** This file will need significant updates to its UPS functions to build the complex request payload required for international shipments, including customs forms and compliance IDs.
- **OrderDetail.jsx:** To be refactored from a large, all-purpose component into a lean "controller." Its primary responsibilities will be to fetch the order data and then render either the DomesticOrderProcessor or InternationalOrderProcessor component based on the customer_shipping_country_iso2 field.
- **InternationalOrderProcessor.jsx (New):** This new component will contain all the UI and logic for handling international orders, including the dynamic forms for customs and compliance information.
- **constants/countries.js (New):** This file will export a static array of country objects { name: "Country Name", code: "CC" } to be used for populating dropdown menus.

6. Summary of Recent Interactions (Current Chat Session)

The following key decisions and designs were finalized during this development session:

- **International Shipping Data Workflow:** A precise, multi-step data lookup process was defined:
    1. Start with the order sku.
    2. Query the existing hpe_part_mappings table to get the option_pn.
    3. Use the option_pn to query the new product_types table to get the product_type.

4. Use the product_type to query the new customs_info table to get the final customs details.

- **Country-Specific Compliance:** A data-driven approach was designed to handle varying compliance requirements (e.g., EORI, IOSS, VAT numbers for Germany). This is managed by the new country_compliance_fields table.
- **Shipper's EIN:** It was determined that all international shipments require the Shipper's EIN, which is the static value 421713620. This value will be stored as a backend configuration constant.
- **Frontend Refactoring:** It was decided to refactor the large OrderDetail.jsx component to improve maintainability. The logic will be split into specialized components for domestic and international order processing.

7. Detailed Plan for Completion & Future Tasks

This is the roadmap for the developer to complete the application.

Phase 1: Backend Setup for Internationalization

1. **Database Migration:**
    - **Action:** Execute the SQL CREATE TABLE commands from Section 4 to create the three new tables (product_types, customs_info, country_compliance_fields) in the PostgreSQL database.
2. **Initial Data Population:**
    - **Action:** Populate the new tables with the necessary data.
        - product_types & customs_info: Populate with the mappings for all relevant products.
        - country_compliance_fields: Populate with the rules for Germany and the global '*' rule for the Shipper's EIN.
3. **Implement New API Endpoint:**
    - **Action:** Create a new API endpoint (e.g., POST /api/order/<order_id>/international-details) in a new blueprints/international.py file.
    - **Logic:** This endpoint must implement the full data workflow from Section 2 and also query for the compliance fields from the country_compliance_fields table.

- **Goal:** The endpoint should return a single, consolidated JSON object with all necessary data for the frontend to render the international shipping forms.

4. **Add Configuration:**
    - **Action:** Add SHIPPER_EIN = "421713620" to the backend application configuration.

Phase 2: Frontend Refactoring & Implementation

1. **Refactor OrderDetail.jsx:**
    - **Action:** Modify OrderDetail.jsx to act as a controller. Move existing domestic processing logic to a new DomesticOrderProcessor.jsx component. Implement the conditional rendering logic to show the correct processor based on the country code.
2. **Build InternationalOrderProcessor.jsx:**
    - **Action:** Create the new InternationalOrderProcessor.jsx component.
    - **Logic:** On load, this component should call the new backend API endpoint to get all international shipping data.
    - **UI:** Render the pre-filled customs information grid and the dynamic compliance ID form based on the API response.
3. **Integrate shipping_service.py:**
    - **Action:** Update the frontend to collect all data from the new international forms and pass it to the backend when the "Process Order" button is clicked. The backend's process_order_route will then need to pass this data to shipping_service.py.
    - **Goal:** The shipping_service.py must be updated to build the correct UPS API request payload for an international shipment, including all customs and compliance data.

Phase 3: Testing and Validation

1. **Unit & Integration Testing:**
    - **Action:** Write tests for the new backend data lookup logic and the frontend components.
2. **End-to-End (E2E) Testing:**

    - **Action:** Perform thorough manual testing of the entire workflow.
    - **Scenarios:**
        - **US Domestic Order:** Verify no changes to existing functionality.
        - **German Order:** Verify all dynamic fields and pre-filled data appear correctly.
        - **Other International Order (e.g., Canada):** Verify only the base international UI and global compliance fields (EIN) appear.
3. **Regression Testing:**
    - **Action:** Test all major existing features (domestic fulfillment, PO generation, QuickBooks IIF files) to ensure no new bugs have been introduced.

Phase 4: Address Remaining Future Tasks

These are items from the original handover report that still need to be addressed after the international shipping feature is complete.

1. **FedEx "Bill RECIPIENT" Authorization:** Follow up with FedEx support to resolve the "SHIPMENT.ACCOUNTNUMBER.UNAUTHORIZED" error.
2. **Shipping Rate Estimates:** Implement functionality to fetch and display shipping rates from both UPS and FedEx APIs before a label is generated.
3. **Address Validation:** Integrate UPS/FedEx address validation APIs to reduce shipping errors.
4. **Automated Testing Framework:** Build out a comprehensive suite of automated unit and integration tests for both the backend and frontend.
5. **Enhanced Reporting:** Expand the reporting capabilities beyond the current daily revenue report.

Development Plan: UPS International Shipping Module

- **Project:** G1 PO App

- **Feature:** UPS International Shipping Automation
- **Date:** May 24, 2025
- **Compiled By:** Gemini Assistant

1. Feature Objective

The primary goal of this module is to integrate UPS international shipping capabilities directly into the order processing workflow. This feature will automate the complex data requirements for international shipments by:

- Automatically looking up and pre-filling customs information (Description, Harmonized Code, Country of Origin) for each item in an order.
- Dynamically generating the required compliance and tax ID forms based on the shipment's destination country.
- Capturing all necessary data to successfully generate a UPS international shipping label and the associated electronic customs documentation (Commercial Invoice) via the UPS API.
- Ensuring this entire UI and logic is presented to the user *only* for orders shipping to non-US destinations.

2. Core Data & Logic Workflow

The backend logic must implement the following multi-step data lookup process to retrieve customs information for each line item:

1. **Start with sku**: For each line item in the order, retrieve the sku.
2. **Get option_pn**: Using the sku, query the existing hpe_part_mappings table to find the corresponding option_pn.
3. **Get product_type**: Using the option_pn from the previous step, query the new product_types table to find the product_type string.
4. **Get Customs Details**: Using the product_type string, query the new customs_info table to retrieve the final customs data: customs_description, harmonized_tariff_code, and default_country_of_origin.

3. Prerequisites

Before development begins, the following database tables must be created in the PostgreSQL database. The developer will need these tables to exist to build and test the API endpoints.

- **product_types**: Stores option_pn to product_type mappings.
- **customs_info**: Stores customs details for each product_type.
- **country_compliance_fields**: Stores rules for which compliance/tax IDs are required for each country.

The approved CREATE TABLE SQL for these tables is provided in the Backend Development section below.

4. Backend Development Tasks (Phase 1)

4.1. Database Schema & Data

1. **Create Tables**: Execute the following SQL commands to create the necessary tables.
    - **Table 1: product_types**

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);
```

    - **Table 2: customs_info**

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);
```

- **Table 3: country_compliance_fields**

SQL

```
CREATE TABLE country_compliance_fields (
  id SERIAL PRIMARY KEY,
  country_iso2 VARCHAR(2) NOT NULL,
  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.';
CREATE INDEX idx_country_iso2 ON country_compliance_fields(country_iso2);
```

2. **Populate Initial Data**: The new tables must be populated with data for testing. At a minimum:
   - Populate product_types and customs_info with data for a few test products.
   - Populate country_compliance_fields with the rules for Germany ('DE') and the global Shipper EIN rule ('*'), as seen in the examples.

4.2. API Endpoint & Logic

1. **Create New API Endpoint**: Implement a new endpoint, e.g., POST /api/order/<order_id>/international-details.
    - This endpoint must perform the complete data lookup workflow detailed in Section 2 for all line items in the order.
    - It must also query the country_compliance_fields table for the order's destination country (using country_iso2 and the '*' wildcard) to get the list of required compliance fields.
    - It should return a consolidated JSON object containing both the pre-filled customs data for each item and the list of required compliance fields for the frontend.
2. **Update shipping_service.py**:
    - Modify the UPS label generation functions in shipping_service.py to accept a new data structure containing all international shipping details (customs items, compliance IDs, etc.).
    - This function must be updated to build the complex JSON payload required by the UPS API for international shipments, including the InternationalForms object for the Commercial Invoice.
3. **Add Configuration**: Add the static Shipper's EIN (421713620) as a configuration variable in the backend (e.g., in .env and app.py) so it can be easily accessed.

5. Frontend Development Tasks (Phase 2)

5.1. File Structure & Refactoring

1. **Create Constants File**: Create src/constants/countries.js and populate it with a comprehensive list of countries and their 2-letter ISO codes.
2. **Refactor OrderDetail.jsx**: Refactor the existing OrderDetail.jsx file to act as a "controller" component.
    - Its primary role will be fetching order data.
    - It will then conditionally render either DomesticOrderProcessor (containing existing logic) or InternationalOrderProcessor based on the order.customer_shipping_country_iso2 field.
3. **Create InternationalOrderProcessor.jsx**: Create this new component file. It will contain all UI and logic specific to international orders.

5.2. InternationalOrderProcessor.jsx Implementation

1. **API Call**: On component mount, call the new backend endpoint (/api/order/<order_id>/international-details) to get all necessary data.
2. **Render Goods Information**: Using the API response, render a grid or list of the order's line items, displaying the pre-filled customs_description and harmonized_tariff_code for each.
3. **Render Dynamic Compliance Form**:
    - Based on the list of required compliance fields from the API, dynamically render a form section.
    - For each required field, display the correct field_label (e.g., "EORI Number").
    - If a field has has_exempt_option: true, render an "Exempt" checkbox next to the input.
    - Pre-fill the Shipper's "EIN" field with the static value and make it read-only.

6. Testing & Validation (Phase 3)

The developer must validate the implementation against the following test cases:

1. **Domestic Order (Regression Test)**: Open an order shipping to the US. **Result:** The UI and functionality must be identical to the current implementation, with no "International" section visible.
2. **German Order (Full Feature Test)**: Open an order shipping to Germany ('DE'). **Result:**
    - The "International Shipment Details" section must be visible.
    - The customs description and harmonized code for each line item must be correctly pre-filled.
    - The compliance form must show inputs for **EIN** (pre-filled), **EORI**, **VAT Number**, and **IOSS Number**.
    - The IOSS Number input must have an "Exempt" checkbox next to it.
3. **Generic International Order (e.g., to Canada)**: Open an order shipping to a country without specific rules defined in the database. **Result:** The compliance form must show only the globally required **EIN** field for the Shipper.

7. Definition of Done

This module is considered complete when:

- All backend tasks in Phase 1 are implemented and unit-tested.
- All frontend tasks in Phase 2 are implemented.
- The frontend successfully communicates with the backend, pre-filling and displaying all required data.
- The data collected from the new UI is successfully passed to the shipping_service.py module.
- All three testing scenarios in Phase 3 pass successfully.

Sources