Project Handover Report: G1 PO App

Date of Report: May 22, 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 (As of End of May 22, 2025)

The application has a significant number of core features implemented. A major backend refactoring introduced Flask Blueprints for better organization and scalability. The

refactored backend has been successfully deployed to Google Cloud Run after resolving various deployment and environment issues.

Key Achievements & Functionalities (Incorporating recent updates):

- **Order Management:** Robust BigCommerce order ingestion, including parsing for third-party shipping accounts and storing billing/shipping cost data.
- **Fulfillment Modes:** Supports Single and Multi-Supplier POs, G1 Onsite Fulfillment, and customer-billed shipments.
- **Document Generation:**
    - PDF generation for Purchase Orders and Packing Slips.
    - Packing slip can be generated in a "blind drop ship" version, omitting company branding and using the customer's billing address as the ship-from on the label. Font was changed to "Eloquia".
- **Shipping Integration:**
    - **UPS:**
        - Functional "Bill Sender" label generation.
        - "Bill Third Party" label generation is now **working correctly** after addressing an issue with the PaymentInformation block (specifically changing PaymentDetails to PaymentInformation and ensuring ShipmentCharge is an object, not an array) in shipping_service.py.
    - **FedEx:** OAuth is working. "Bill SENDER" labels have been tested in production (test-marked). "Bill RECIPIENT" functionality is implemented in the code but currently encounters a "SHIPMENT.ACCOUNTNUMBER.UNAUTHORIZED" error from the FedEx API, which requires FedEx support/configuration to resolve for production accounts. FedEx production API credentials are set up and selected based on the environment.
- **Communication:** Email communications for POs to suppliers via Postmark.
- **BigCommerce Updates:** Integration for order data retrieval and updating order status/shipment details.
- **QuickBooks Integration:** Extensive QuickBooks Desktop IIF file generation for Purchase Orders, Sales Orders (as Invoices), and corresponding Payments.

Includes US Central Time conversion and specific formatting requirements. The IIF generator was updated to handle SKUs with underscores for product mapping lookups.

- **Authentication:** Firebase Authentication (Google Sign-In with an isApproved custom claim) is implemented for user access control.
- **Frontend (OrderDetail.jsx):**
    - Features a Dashboard, a detailed OrderDetail.jsx page for order processing, and a Utilities section. Dark mode CSS is implemented.
    - OrderDetail.jsx has been updated to include FedEx shipping options, UI for "Bill to Customer's FedEx Account," and a "Blind Drop Ship?" option.
    - The "Blind Drop Ship?" checkbox has been renamed to "Branding" dropdown with options: "Global One Technology Branding" and "Blind Ship (Generic Packing Slip)".
    - This "Branding" dropdown has been relocated to be the last input just above the "Process Order" button in both single-supplier and multi-supplier views within OrderDetail.jsx.
    - Layout adjustments for the "Shipment Information" section and "Process PO" button centering were made.
    - An Uncaught ReferenceError: currentSelectedShippingOption is not defined in OrderDetail.jsx was addressed by making the getSelectedShippingOption function more robust and ensuring currentSelectedShippingOption is correctly derived.
- **Backend Refactoring:** API route logic was moved from the main app.py into Flask Blueprints for improved modularity. app.py now handles core setup, shared resources, and blueprint registration.

---

3. Technology Stack & Project Layout

(Largely as per "Handover Report 20250520b.pdf", with updates reflected)

- **Backend (order-processing-app directory):**
    - Language/Framework: Python 3.9+, Flask
    - WSGI Server: Gunicorn

    - Key Libraries: SQLAlchemy, google-cloud-sql-connector[pg8000], requests, python-dotenv, ReportLab, Pillow, google-cloud-storage, firebase-admin, postmarker, pytz.
    - Updated Directory Structure:
        - app.py: Main Flask app, shared resources, blueprint registration.
        - blueprints/: Contains individual Python files for each blueprint (e.g., orders.py, suppliers.py, quickbooks.py, hpe_mappings.py, reports.py, utils_routes.py).
        - Service Modules: document_generator.py, email_service.py, iif_generator.py, shipping_service.py.
        - Configuration: .env, requirements.txt, Dockerfile, entrypoint.sh.
- **Frontend (order-processing-app-frontend directory):**
    - Framework/Library: React (Vite build tool)
    - Routing: react-router-dom
    - Authentication: Firebase Client SDK
    - Styling: Standard CSS, component-specific CSS (e.g., OrderDetail.css, App.css). OrderDetail.css was implicitly modified to accommodate layout changes for the branding dropdown.
- **Database:** PostgreSQL on Google Cloud SQL.
    - A new column customer_shipping_country_iso2 VARCHAR(2) was added to the orders table.
- **Cloud Platform (GCP):** Cloud Run, Cloud SQL, Artifact Registry, Secret Manager, Cloud Storage, Cloud Scheduler, Cloud Logging, IAM.
- **Authentication:** Firebase Authentication (Google Sign-In, Custom Claims: isApproved: true).
- **External APIs:** BigCommerce API, UPS API (OAuth 2.0), FedEx API (OAuth 2.0), Postmark API.

---

4. Key Code Files & Their Roles

(Incorporating updates from recent changes)

- **Backend (order-processing-app):**
    - app.py: Core application setup, initializes Flask, configurations, database, GCS, Firebase Admin SDK. Defines shared helpers. Registers all blueprints. Contains root routes.
    - blueprints/orders.py: Handles order listing, details, status updates, BigCommerce order ingestion, and the main order processing logic (process_order_route). This route includes PO creation, document generation calls, shipping label generation calls, and BigCommerce updates. It was updated to include FedEx label generation logic, handle a "Blind Drop Ship" flag (changing ship-from address and packing slip type), and enforce that supplier emails are only sent if all three documents (PO, Packing Slip, Label - if label attempted) are generated.
    - blueprints/suppliers.py: CRUD operations for suppliers.
    - blueprints/hpe_mappings.py: Manages HPE Part Number to PO Description mappings and SKU lookups.
    - blueprints/quickbooks.py: Endpoints for triggering QuickBooks IIF file generation.
    - blueprints/reports.py: For generating reports like the daily revenue report.
    - blueprints/utils_routes.py: Utility endpoints, e.g., standalone UPS label generator.
    - shipping_service.py:
        - Contains logic for interacting with UPS and FedEx APIs to get shipping rates (future) and generate shipping labels. Handles OAuth for both. FedEx logic selects production credentials based on FEDEX_API_ENVIRONMENT.
        - **Recently Updated:** The generate_ups_label_raw function was modified to correctly implement "Bill Third Party" functionality. This involved changing the PaymentDetails key to PaymentInformation and ensuring the ShipmentCharge sub-object is a direct object rather than an array, aligning with UPS API specifications for the v2409 API version. This resolved the issue with third-party UPS billing.

- document_generator.py: Generates PDF documents (Purchase Orders, Packing Slips). Updated to accept is_blind_slip and custom_ship_from_address parameters to generate generic packing slips and conditionally omit the company logo/footer for blind drop shipments. The packing slip font was changed to "Eloquia".
- email_service.py: Handles sending emails (e.g., POs to suppliers) via Postmark.
- iif_generator.py: Contains all logic for generating IIF files for QuickBooks Desktop. Updated to handle SKUs with underscores.
- Dockerfile & entrypoint.sh: Configure the Docker container and how Gunicorn runs the Flask app in Cloud Run.

- **Frontend (order-processing-app-frontend):**
  - OrderDetail.jsx:
    - The primary component for viewing and processing individual orders.
    - Saw significant changes for FedEx shipping methods, UI for "Bill to Customer's FedEx Account," and the "Blind Drop Ship?" option.
    - Layout adjustments for "Shipment Information" and "Process PO" button centering.
    - Fixes for the "page goes blank" error related to currentSelectedShippingOption.
    - **Recently Updated:**
      - The "Blind Drop Ship?" checkbox was converted into a "Branding" dropdown.
      - The options for this dropdown were changed from "G1 Branding" / "Blind/No Branding" to "Global One Technology Branding" / "Blind Ship (Generic Packing Slip)".
      - This "Branding" dropdown was moved to be the last input element just before the main "Process Order" button in both single-supplier and multi-supplier fulfillment views.

- OrderDetail.css: Styles for OrderDetail.jsx, updated to support dark mode and new layout adjustments. "Empty ruleset" linter warnings were addressed. Implicitly affected by the branding dropdown relocation.
- App.css: Global styles, including dark mode variables and base styling.

---

5. Database Schema Notes & Key Tables

(Refer to "Handover Report 20250520b.pdf," Section 5, pages 13-15 for the original detailed schema)

- **orders table:**
  - Added customer_shipping_country_iso2 VARCHAR(2): Stores the 2-letter ISO code for the shipping country. This was to resolve an ingestion error.
  - The customer_shipping_country column should store the full country name for display purposes.
  - QuickBooks sync status columns (qb_po_sync_status, qb_sales_order_sync_status, etc.) are crucial for IIF generation logic.
  - Ensure customer_ups_account_zipcode is being correctly populated during ingestion if it's intended to be used for third-party UPS billing (as seen as relevant in shipping_service.py and orders.py).

---

6. Developer Setup & Environment

(Refer to "Handover Report 20250520b.pdf," Section 6, pages 16-18 for base setup)

- **Backend:**
  - Python 3.9+ virtual environment is essential. Resolved "Fatal error in launcher" on Windows by recreating venv.
  - PowerShell Execution Policy: May need Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser.
  - Gunicorn on Windows: Use Flask's dev server (flask run) or Waitress for local WSGI testing due to fcntl error.
  - Line Endings: Ensure .sh files use LF (Unix) line endings.

- **Frontend:** Standard npm install and npm run dev/npm run build.
- **Deployment:**
    - Backend to Cloud Run via Docker: Use gcloud auth configure-docker, docker build, docker tag, docker push, and gcloud run deploy with appropriate environment variables and secrets.
    - Frontend to Firebase Hosting: npm run build, then firebase deploy --only hosting.
- **Crucial Environment Variables:** Extensive list in the previous report and user's deploy scripts. For frontend, VITE_API_BASE_URL and Firebase SDK config are key. Ensure all SHIP_FROM_* variables, database credentials, API keys (BigCommerce, UPS, FedEx, Postmark), and GCS bucket names are correctly set in .env for local development and as secrets/env vars in Cloud Run.

---

7. Summary of Interactions (May 21-22, 2025 - per original PDF)

This section summarizes the work completed *prior* to the current interactive session, as documented in the "Handover Report 20250522.pdf".

- **FedEx Integration Advancement:**
    - Reviewed existing FedEx "Bill Recipient" logic in shipping_service.py.
    - Updated OrderDetail.jsx for FedEx methods, UI for "Bill to Customer's FedEx Account," and account number input.
    - Modified blueprints/orders.py (process_order_route) to handle carrier info (UPS/FedEx), FedEx billing details, conditionally call label functions, and pass necessary flags.
    - Diagnosed FedEx API error "SHIPMENT.ACCOUNTNUMBER.UNAUTHORIZED" for "Bill Recipient," confirming it requires FedEx account configuration.
- **QuickBooks IIF Generator Update:**
    - Modified iif_generator.py to handle SKUs with underscores by attempting a lookup on the part of the SKU after the last underscore if a direct lookup fails.
- **Order Processing Enhancements:**

    - Modified blueprints/orders.py (process_order_route) to prevent sending supplier emails and updating BigCommerce if not all three documents (PO, Packing Slip, Label - if attempted) are generated for a supplier PO where a label was attempted; an error is raised instead.

- **Blind Drop Ship Functionality:**
    - OrderDetail.jsx: Added a global "Blind Drop Ship?" checkbox. (Note: This has since been updated in the *current* chat session).
    - blueprints/orders.py: Updated process_order_route to receive the is_blind_drop_ship flag, use the order's billing address as ship-from if true, and pass flags to document_generator.py.
    - document_generator.py: Modified generate_packing_slip_pdf to accept is_blind_slip and custom_ship_from_address, omit company logo/footer, and use generic/custom ship-from details if is_blind_slip is true.
- **Font Change:** Updated document_generator.py to use "Eloquia" fonts for the Packing Slip.
- **Frontend Layout Adjustments (OrderDetail.jsx & OrderDetail.css):**
    - Addressed layout requests for "Shipment Information" section elements for desktop and mobile views.
    - Adjusted CSS to center the "Process PO" button.
- **Database Schema Update:** Added customer_shipping_country_iso2 VARCHAR(2) to the orders table.
- **Critical Bug Fix (OrderDetail.jsx):** Resolved an Uncaught ReferenceError: currentSelectedShippingOption is not defined by making the getSelectedShippingOption helper function more robust.

---

8. Summary of Interactions (Latter Part of May 22, 2025 - Current Chat Session)

This section details the work completed during the *current* interactive session with the Gemini Assistant, building upon the state described above.

- **UPS "Bill Third Party" Shipping Fix:**
    - **Issue:** Troubleshooting failures with shipping via "Bill Third Party" on a customer's UPS account.

- **Analysis:** Compared the existing generate_ups_label_raw function in shipping_service.py with the UPS API documentation/example for "BillThirdParty" shipments (API version v2409).
- **Key Findings & Resolution:**
  - The implementation was using "PaymentDetails" instead of the more standard "PaymentInformation" for the payment block. This was changed.
  - The ShipmentCharge element within the payment block was being sent as an array [{...}] instead of a direct object {...}. This was corrected.
  - The Type = "01" for ShipmentCharge was confirmed to be correctly implemented.
- **Outcome:** These changes were applied to shipping_service.py, and the user confirmed that "Bill Third Party" UPS shipping is **now working**.

- **OrderDetail.jsx UI Enhancements:**
  - **Branding Dropdown Labels Updated:**
    - The option "G1 Branding" was changed to "Global One Technology Branding".
    - The option "Blind/No Branding" (previously "Blind Drop Ship?") was changed to "Blind Ship (Generic Packing Slip)".
    - These changes apply to the branding selection in single-supplier, G1 Onsite, and multi-supplier modes.
  - **Branding Dropdown Relocation:**
    - **Single Supplier View:** The "Branding" dropdown (with its label) was moved from within the "Shipment Information" card to its own new section, positioned directly above the "PROCESS SINGLE SUPPLIER PO" button.
    - **Multi-Supplier View:** The entire "Global Options" card (which contains the "Branding" dropdown) was moved to be positioned directly above the "PROCESS ALL ASSIGNED POs" button.

---

9. Detailed Plan for Completion & Future Tasks

This plan integrates tasks from the previous "Handover Report 20250520b.pdf" with new items, progress from all recent sessions, and considerations for future development.

Phase 0: Immediate Post-Handover Validation & Critical Fixes (Some of these were in the previous report and remain critical)

1. **Thorough API Testing (Critical):**
    - **Action:** Systematically test all API endpoints (GET, POST, PUT for orders, suppliers, HPE mappings; POST for QuickBooks triggers, label generation, order processing) using the deployed frontend.
    - **Focus:**
        - Confirm the backend refactoring into blueprints and recent changes (FedEx Bill Sender/Recipient (pending auth fix), **UPS Bill Third Party (now working)**, Blind Drop Ship, IIF underscore fix, strict document check) work as expected without regressions.
        - Verify data handling, document generation (all types, including blind ship versions), and external API calls (BigCommerce, Postmark, UPS, FedEx).
        - Test the recent OrderDetail.jsx branding dropdown changes and their impact on data passed to process_order_route.
    - **Goal:** Confirm backend and frontend stability, data integrity, and that all recent fixes and features operate correctly.
2. **Monitor Cloud Run Logs (Ongoing):**
    - **Action:** While testing, continuously monitor Cloud Run logs for the deployed backend service.
    - **Goal:** Ensure application stability, identify any new runtime errors, unexpected behavior, or performance issues.
3. **Resolve Firebase auth/quota-exceeded Error (Critical if still occurring):**
    - **Action:** Check Google Cloud Console for "Identity Platform" / "Firebase Authentication" quotas.
    - **Goal:** If quotas are the issue, upgrade the Firebase project to the "Blaze" (pay-as-you-go) plan to ensure reliable authentication.

4. **Review & Tune Gunicorn Production Settings:**
    - **Action:** Current entrypoint.sh uses debug settings. After testing confirms stability, adjust for production (e.g., increase threads/workers, set log-level to info or warning).
    - **Goal:** Optimized and stable Gunicorn configuration.
5. **Frontend Bug Squashing (Post currentSelectedShippingOption fix & recent UI changes):**
    - **Action:** Thoroughly test all supplier/mode selections in OrderDetail.jsx to ensure no "blank page" errors occur and that the new branding dropdown logic works correctly across all modes. Check browser console for any other JS errors.
    - **Goal:** Stable OrderDetail.jsx component during dynamic state changes.

Phase 1: Frontend UI/UX Refinement (OrderDetail.jsx and other components) (Some items from original report, potentially impacted by recent changes)

1. **Finalize Shipment Information Layout (Desktop & Mobile):**
    - **Action:** Review and implement the latest discussed CSS and JSX for the "Method", "Weight (lbs)" fields to achieve the desired layout in both Single Supplier and Multi-Supplier (per PO) views.
        - Desktop: "Method" on its own row. "Weight (lbs)" label + input on the next row. (Note: The original report mentioned "Blind Ship?" here, which has now been moved).
        - Mobile: Ensure these elements stack cleanly and logically.
    - **Goal:** User-approved layout for shipment inputs.
2. **Item Purchase Grid Layout - "Unit Cost" (Desktop - Single Supplier View):**
    - **Action:** Adjust CSS for .item-row or .qty-cost-row within the "Items to Purchase" section to ensure the "Unit Cost" input field appears directly under its "Unit Cost" label, similar to how "Qty" is structured.
    - **Goal:** Consistent and clear layout for item entry.
3. **Mobile View - Status Alignment:**

- **Action:** Verify the order status badge aligns correctly with the order title on mobile views. Adjust CSS if necessary.
- **Goal:** Improved mobile layout.

4. **Desktop View - Address Duplicate Status / Layout Transformation:**
    - **Action:** Clarify with the user which "table" layouts need conversion to cards and where any duplicate statuses appear. Implement a responsive multi-column card layout.
    - **Goal:** Modernized and responsive UI for list views.
5. **Styling for Relocated Branding Dropdown:**
    - **Action:** Ensure the newly relocated "Branding" dropdown (and its "Global Options" container in multi-supplier mode) is styled correctly and consistently with the rest of the form, especially concerning label alignment and spacing. Adjust OrderDetail.css as needed.
    - **Goal:** Visually integrated and clean UI for the branding options.

Phase 2: Core Functionality Enhancements & New Features

1. **UPS International Shipping:**
    - **Action:**
        - Research UPS API requirements for international shipments (e.g., customs forms like Commercial Invoice, commodity information, duties and taxes, EORI numbers, etc.).
        - Update shipping_service.py to include new fields and logic in the UPS API request payload for international shipments. This may involve:
            - Adding parameters to generate_ups_label_raw (and subsequently generate_ups_label) for customs information.
            - Modifying the JSON payload to include ShipmentServiceOptions (for InternationalForms like CommercialInvoice), PackageServiceOptions if needed.
            - Handling SoldTo address details if different from ShipTo.
            - Managing Commodity details for each item being shipped.

        - Update blueprints/orders.py (process_order_route) to gather and pass necessary international shipping data from the frontend/database to shipping_service.py.
        - Update OrderDetail.jsx to include UI elements for capturing international shipping details (e.g., item descriptions for customs, harmonized codes (optional), country of origin, declared value per item, reason for export). This will likely involve significant form changes when an international shipment is detected or selected.
        - Update document_generator.py if any local generation of international forms (e.g., a pro-forma commercial invoice if UPS doesn't provide a sufficient one via API) is required, or to ensure packing slips contain all necessary info.
        - Consider how to store and manage customs-related product information (e.g., harmonized codes, country of manufacture) in the database, potentially in a new table or by extending order_line_items or hpe_mappings.
    - **Goal:** Enable generation of UPS shipping labels and required documentation for international destinations.
2. **FedEx "Bill RECIPIENT" Authorization Fix:**
    - **Action:** Follow up with FedEx support to understand the cause of the "SHIPMENT.ACCOUNTNUMBER.UNAUTHORIZED" error and what configuration changes are needed on the FedEx account(s) to allow billing to a recipient's account.
    - **Goal:** Fully functional FedEx "Bill RECIPIENT" label generation.
3. **Shipping Rate Estimates:**
    - **Action:** Implement functionality in shipping_service.py to fetch shipping rates from UPS and FedEx APIs. This will involve new API calls and payload structures.
    - Update OrderDetail.jsx to display rate estimates, potentially allowing users to choose services based on cost and speed.
    - **Goal:** Provide users with shipping cost estimates before finalizing shipments.
4. **Address Validation:**

- **Action:** Integrate UPS/FedEx address validation APIs into shipping_service.py.
- Provide feedback in OrderDetail.jsx if an address is potentially problematic.
- **Goal:** Reduce shipping errors and returns due to incorrect addresses.

5. **Enhanced Error Handling & User Feedback:**
   - **Action:** Review all API interactions (BigCommerce, UPS, FedEx, Postmark) for more robust error handling and clearer feedback to the user in the frontend.
   - **Goal:** Improve application resilience and user experience when issues occur.
6. **Automated Tests (Backend & Frontend):**
   - **Action:** Develop unit and integration tests for backend (Python/Flask) and frontend (React/Vitest or Jest).
   - **Goal:** Ensure code quality, prevent regressions, and facilitate future development.
7. **Consolidate Frontend State Management (If needed):**
   - **Action:** Evaluate if a more robust state management solution (e.g., Redux Toolkit, Zustand) is needed for OrderDetail.jsx or globally as complexity grows. Currently, useState and useCallback are used extensively.
   - **Goal:** Maintainable and scalable frontend architecture.
8. **Comprehensive User Documentation/Guide:**
   - **Action:** Create a user guide explaining how to use all features of the G1 PO App.
   - **Goal:** Enable users to effectively utilize the application.

Phase 3: Advanced Features & Integrations

1. **Partial Shipment Tracking within a Single Order:**
   - **Action:** If an order is split into multiple POs/shipments (as supported by Multi-Supplier mode), enhance the UI to clearly track each part of the shipment individually.
   - **Goal:** Better visibility for complex orders.

2. **Inventory Management Aspects (Future Consideration):**
    - **Action:** Explore if basic inventory tracking for G1-stocked items is desired.
    - **Goal:** Potential expansion of app capabilities.
3. **More Sophisticated Reporting:**
    - **Action:** Expand on the current daily revenue report with more customizable and detailed reporting features.
    - **Goal:** Provide better business intelligence.

This detailed plan should provide a clear roadmap for the continued development and completion of the G1 PO App.

Sources