G1 PO App - Comprehensive Handover Report & Next Steps

Date: May 13, 2025 (Reflecting current progress and chat session ending May 12, 2025 CDT)

Prepared For: Incoming Developer

Prepared By: Mark (via collaboration with AI Assistant Gemini)

Version: 2.1 (Supersedes report dated May 13, 2025, from Canvas ID g1_po_app_handover_report_final_20250513)

1. Introduction

This document provides an updated and comprehensive overview and handover for the "G1 PO App" project. It consolidates information from previous handover reports with all subsequent development progress. This includes the implementation and extensive debugging of the daily IIF file generation for QuickBooks Desktop 2020, critical updates to backend services (app.py, shipping_service.py), and the successful setup and initial testing of the automated task triggering via Google Cloud Scheduler. The goal remains to provide a clear and detailed guide for the incoming developer to understand the project's current state, architecture, codebase, deployment, and the remaining tasks to achieve the Minimum Viable Product (MVP) and future enhancements.

2. Project Overview (Recap)

- **Goal:** To develop a web application ("G1 PO App") that automates the purchase order (PO) and shipment process for drop-shipped items originating from BigCommerce orders.
- **Core Functionality (Original Intent & Current Implementation):**
    - Ingest relevant orders from BigCommerce (filtered by status, e.g., "Awaiting Fulfillment").
    - Provide a web interface for users to review these orders.
    - Allow users to trigger processing for selected orders.
    - Generate Purchase Orders (PDFs) using specific HPE Option PNs mapped from original BigCommerce SKUs.
    - Generate Packing Slips (PDFs).
    - Generate UPS Shipping Labels (PDFs) via UPS API (OAuth 2.0).

    - Email generated PO, Packing Slip, and Label to the selected supplier via Postmark API.
    - Upload generated documents (PO, Packing Slip, Label) to Google Cloud Storage (GCS).
    - Update order status and add tracking information in BigCommerce via its API.
    - Update local application database status for processed orders (orders and purchase_orders tables).
    - Provide a web interface (React/Vite) for viewing/filtering orders, viewing details, interactively preparing POs, and triggering processing.
    - **COMPLETED & VERIFIED:** Generate a daily IIF (Intuit Interchange Format) file for Purchase Orders processed on the previous day, suitable for import into QuickBooks Desktop 2020. This IIF file is emailed to a designated address.
    - (Future) Manage suppliers and product mappings more extensively via the UI.

3. Current Status (As of May 13, 2025)

The application is largely functional, with core backend processing deployed and the IIF generation feature now operational and tested successfully for import into QuickBooks Desktop 2020.

Backend (Python/Flask - app.py and service modules):

- **Deployment:** Dockerized and deployed to Google Cloud Run.
    - Service Name: g1-po-app-backend-service
    - Region: us-central1
    - URL: https://g1-po-app-backend-service-992027428168.us-central1.run.app
- **Database:** Connected to a PostgreSQL instance on Google Cloud SQL.
    - Instance Connection Name: order-processing-app-458900:us-central1:order-app-db
    - Connection managed via Cloud SQL Auth Proxy sidecar in Cloud Run.
- **Secrets Management:** All sensitive credentials managed via Google Cloud Secret Manager.

- **Order Ingestion (/api/ingest_orders - POST):**
    - Functional. Fetches orders from BigCommerce, saves/updates in local DB.
    - Correctly determines international status and saves customer notes.
    - Fetches total_tax from BigCommerce and stores it in orders.bigcommerce_order_tax. Stores price_ex_tax in order_line_items.sale_price.
- **Order Retrieval (/api/orders, /api/orders/<id> - GET):** Functional.
- **Order Processing (/api/orders/<id>/process - POST):**
    - Core workflow is functional: PO PDF, Packing Slip PDF, UPS Label PDF generation (via GIF conversion).
    - Uploads documents to GCS.
    - Emails documents to supplier (Postmark).
    - Sends structured PO data email for QuickBooks (separate from IIF).
    - Updates BigCommerce order status and tracking.
    - Updates local orders table status to "Processed".
    - **Critical Update:** Updates local purchase_orders table status to "SENT_TO_SUPPLIER" after successful supplier email. This is the trigger for IIF generation. (Implemented in app.py version corresponding to Canvas ID app_py_po_status_update).
    - **Bug Fix:** Corrected po_line_items insert to use the correct sku column name instead of item_sku. (Implemented in app.py version corresponding to Canvas ID app_py_diagnostic_print and verified in subsequent versions).
- **IIF Generation (iif_generator.py):**
    - **Functionality:** Generates a daily IIF file for Purchase Orders. Successfully tested for import into QuickBooks Desktop 2020.
    - **Trigger:** An API endpoint (/api/tasks/trigger-daily-iif in app.py) calls the generation function. This endpoint is triggered by a Cloud Scheduler job.
    - **Logic (Final Working Version - corresponds to Canvas ID iif_generator_scenario_a with title "iif_generator.py - Production Date**

Logic" after confirming the fix from "iif_generator.py - Full QB Desktop PO Format"):

            - Processes purchase_orders records with status = 'SENT_TO_SUPPLIER' and po_date matching the previous day.
            - Uses the "Purchase Orders" non-posting account for TRNS.ACCNT and a negative total amount for TRNS.AMOUNT.
            - Item SPL lines use "Cost of Goods Sold" (or configured default) and positive amounts.
            - Maps po_line_items.sku (Option PN) to QuickBooks item names via the qb_product_mapping table.
            - Correctly handles and preserves double quotes (") and pipe (|) characters in item names/descriptions by adjusting the sanitize_field function (double quotes are kept, pipe characters are kept as per latest request).
            - Includes payment_instructions from purchase_orders as a zero-amount SPL line memo.
            - Includes a "Fulfillment of G1 Order #[BigCommerce Order ID]" note as a zero-amount SPL line memo.
            - Outputs blank values for TERMS and SHIPVIA on the TRNS line to prevent QuickBooks list validation errors during import.
            - Includes Ship-To address details on the TRNS line.
            - Uses Windows-style line endings (\r\n) and robust field sanitization.
            - The sanitize_field function was iteratively refined to handle newlines, tabs, and to correctly preserve necessary characters like " and | based on QuickBooks import behavior.
    - **Emailing:** The generated IIF file is emailed to the address specified in QUICKBOOKS_EMAIL_RECIPIENT.
- **API Endpoint for IIF Task (/api/tasks/trigger-daily-iif - POST):**
    - Added to app.py (version corresponding to Canvas ID app_py_trigger_iif_endpoint).
    - Successfully tested.

- **Other Endpoints:** Status Update, SKU/Description Lookup, Supplier & Product Mapping CRUD endpoints are functional.
- **CORS:** Configured in app.py.

Frontend (React/Vite):

- **Deployment:** Deployed to Firebase Hosting. URL: https://g1-po-app-77790.web.app.
- **Functionality:** As per previous reports. No recent changes in this chat session.
- **Font Issue:** Eloquia font rendering issue on the deployed site remains pending.

Database (PostgreSQL on Cloud SQL):

- Core tables are defined and functional.
- qb_product_mapping table (columns: option_pn (PK), qb_item_name) created and populated. This table is critical for mapping internal SKUs/Option PNs to the exact item names required by QuickBooks Desktop.
- purchase_orders.status column is now actively used and updated.

Integrations:

- **UPS API:** Functional.
- **BigCommerce API:** Functional.
    - shipping_service.py's update_bigcommerce_order function was updated to correctly accept and use the shipped_status_id parameter.
    - shipping_service.py's generate_ups_label_raw function was updated to fix a TypeError related to transaction_identifier concatenation (latest version in Canvas ID shipping_service_py_full with title "shipping_service.py - Verify Version & Fix TypeError").
- **Postmark API:** Functional for all email types.
- **Google Cloud Storage (GCS):** Functional.

Scheduling (IIF Generation):

- **Google Cloud Scheduler:** Job configured to call /api/tasks/trigger-daily-iif via HTTP POST.

    - **Frequency:** Daily at 11:45 PM (user to confirm and set desired timezone in GCP).
    - **Authentication:** OIDC token using a dedicated service account with "Cloud Run Invoker" role.
    - **Status:** Successfully tested after correcting the target URL typo and ensuring the endpoint was deployed.

4. Technology Stack (Recap)

- **Backend:** Python 3.9+, Flask, SQLAlchemy, pg8000, requests, google-cloud-sql-connector, google-cloud-storage, reportlab, Pillow, Flask-CORS, postmarker, python-dotenv, re.
- **Frontend:** React, Vite, react-router-dom, axios.
- **Database:** PostgreSQL (on Google Cloud SQL).
- **Cloud Platform (GCP):** Cloud Run, Cloud SQL, Artifact Registry, Secret Manager, GCS, Cloud Scheduler.
- **Firebase Hosting:** For frontend.
- **APIs:** UPS, BigCommerce, Postmark.

5. Codebase Structure (Key Files & Recent Changes)

- **Backend (order-processing-app directory):**
    - app.py: (Latest version corresponds to Canvas ID app_py_diagnostic_print or the version immediately preceding it that was confirmed deployed, e.g., app_py_trigger_iif_endpoint after ensuring the po_line_items.sku fix was in the deployed revision g1-po-app-backend-service-00056-48w or newer).
        - Added /api/tasks/trigger-daily-iif endpoint.
        - Updated process_order to set purchase_orders.status to "SENT_TO_SUPPLIER".
        - Corrected po_line_items insert in process_order to use sku column.
    - shipping_service.py: (Latest version corresponds to Canvas ID shipping_service_py_full with title "shipping_service.py - Verify Version & Fix TypeError").

        - update_bigcommerce_order function now correctly handles shipped_status_id.
        - generate_ups_label_raw function fixed TypeError for transaction_identifier.
    - iif_generator.py: (Latest version corresponds to Canvas ID iif_generator_scenario_a with title "iif_generator.py - Production Date Logic", which incorporates all fixes for QB Desktop 2020 import, including supplier name stripping and fulfillment note).
        - Generates IIF for QuickBooks Desktop 2020 Purchase Orders.
        - Uses "Purchase Orders" non-posting account, negative TRNS amount.
        - Handles item mapping, preserves " and | in item names/descriptions.
        - Includes payment instructions and fulfillment notes as separate zero-amount SPL lines.
        - Outputs blank TERMS and SHIPVIA.
        - Uses Windows-style line endings and robust field sanitization.
    - email_service.py: send_iif_batch_email updated for HTML warnings.
    - Dockerfile, entrypoint.sh, requirements.txt.

6. API Integrations & Credentials

- Managed in Google Cloud Secret Manager for deployed services.
- Local development uses .env.

7. Key Features Implemented / Progress Details (Summary)

- Core order processing workflow: **COMPLETED & VERIFIED.**
- Daily IIF Generation for QuickBooks Desktop POs: **COMPLETED & VERIFIED.**
- Automated triggering of IIF generation via Cloud Scheduler: **SETUP & INITIAL VERIFICATION COMPLETE.**
- All critical bug fixes in app.py and shipping_service.py related to database operations and external API calls: **COMPLETED.**

8. Critical Next Steps / Remaining MVP Tasks (Updated Plan)

1. **Finalize and Monitor Cloud Scheduler for IIF Generation:**
    - **Action:** Confirm the Cloud Scheduler job for /api/tasks/trigger-daily-iif is running correctly on its production schedule (e.g., 11:45 PM daily in the correct timezone).
    - **Action:** Monitor Cloud Run logs and email delivery for the first few scheduled runs (e.g., for 2-3 days) to ensure stability and correctness of the IIF files and their import into QuickBooks.
    - **Action:** The iif_generator.py is currently set for production date logic (yesterday's POs). This is correct.
2. **PO Export Feature (Excel - Backend & Frontend):**
    - **Status:** PENDING.
    - **Goal:**
        - Backend: Create GET /api/exports/pos endpoint to generate and return an Excel (.xlsx) file summarizing Purchase Order details (columns to be defined based on user needs, e.g., PO Number, Date, Supplier, Item SKU, Item Description, Qty, Cost, Total).
        - Frontend: Add a button/link on Dashboard.jsx (perhaps with date range filters) to trigger this export and download the file.
    - **Technical Details:**
        - Backend: Use Flask's send_file or make_response. Library: openpyxl is recommended for .xlsx generation.
        - SQL Query: Will need to join purchase_orders, po_line_items, and suppliers.
        - Frontend: Standard API call (fetch or axios) to the new endpoint. Handle the file download response (e.g., using Blob and URL.createObjectURL).
3. **Comprehensive End-to-End Testing (Beyond IIF):**
    - **Status:** PARTIALLY DONE. FORMAL & FULL TESTING PENDING.
    - **Action Items:**

        - Test full order processing flows (BigCommerce order -> G1 PO App Ingestion -> UI Review -> Process Order button -> PDF generation -> GCS Upload -> Supplier Email -> BigCommerce Status Update) for various scenarios (direct HPE mapping, underscore fallback, manual SKU/desc overrides by user in UI if that feature is built).
        - Test edge cases for all API integrations (UPS, BigCommerce, Postmark).
        - Thoroughly test all frontend interactions, form submissions, and error handling displays.
        - User Acceptance Testing (UAT) with the end-user(s).

4. **Data Integrity for QuickBooks Lists (TERMS and SHIPVIA - Future Enhancement):**
    - **Status:** Currently N/A as TERMS and SHIPVIA are intentionally blanked in the IIF file to ensure successful import.
    - **Goal (If desired later):** Populate TERMS and SHIPVIA fields in the IIF.
    - **Action:**
        - Ensure data in suppliers.payment_terms exactly matches entries in the QuickBooks "Terms List".
        - Ensure data for shipping methods (likely from orders.customer_shipping_method) is mapped or truncated to exactly match entries in the QuickBooks "Ship Via List" (max 15 characters).
        - This might involve adding dropdowns in the frontend for supplier/order creation/editing that are populated from valid QuickBooks lists, or a backend mapping mechanism.
5. **Postmark - External Domain Sending Approval:**
    - **Status:** PENDING VERIFICATION.
    - **Action Item:** Ensure the Postmark account is fully approved for sending emails to external supplier domains. This typically requires DNS changes (DKIM, SPF, Custom Return-Path) for the sending domain used in EMAIL_SENDER_ADDRESS.
6. **Frontend - Eloquia Font Issue:**
    - **Status:** PENDING.

- **Action Items:** Investigate and resolve the font rendering issue on the deployed Firebase site. Check CSS font-face rules, file paths, network requests for font files in browser dev tools, and computed styles.

7. **Cloudflare DNS for g1po.com:**
   - **Status:** PENDING.
   - **Action Items:**
     - In Firebase Hosting console, add g1po.com as a custom domain.
     - Firebase will provide DNS records (A, TXT). Add/update these in Cloudflare DNS settings for g1po.com. Set A records to "DNS Only" initially for Firebase verification.
     - Wait for Firebase to verify and provision SSL.
     - Once https://g1po.com works, consider changing A records in Cloudflare to "Proxied".
     - (Optional but Recommended for API) Set up api.g1po.com:
       - Add CNAME api in Cloudflare pointing to the Cloud Run backend URL (g1-po-app-backend-service-....run.app), "Proxied".
       - Map api.g1po.com as a custom domain in the Cloud Run service settings in GCP.
       - Update VITE_API_BASE_URL in frontend's .env.production to https://api.g1po.com/api, rebuild, and redeploy frontend.
8. **Security Hardening & Best Practices:**
   - **Status:** PENDING.
   - **Action Items:**
     - **Cloud Run Authentication:** Transition the main Cloud Run service from --allow-unauthenticated. Secure all endpoints. The /api/tasks/trigger-daily-iif endpoint is already set up to use OIDC with Cloud Scheduler, which is good. The /api/ingest_orders (if still triggered externally and unauthenticated) should also be secured, perhaps with an API key or OIDC if called by another service. Frontend

        calls will need to handle authentication (e.g., Firebase Authentication, passing JWTs to the backend).

        - **IAM Least Privilege:** Ensure the service account used by Cloud Run and any other service accounts have only the minimum necessary IAM roles.
        - **Input Validation:** Thoroughly review all backend API endpoints for robust input validation on all incoming data.
        - **Error Handling & Logging:** Enhance user-facing error messages on the frontend. Implement more structured and detailed logging on the backend (e.g., using Python's logging module with different levels, potentially sending logs to Google Cloud Logging more effectively).

9. **Supplier and Product (HPE Mapping & QB Mapping) Management UI:**
    - **Status:** PENDING (Basic placeholders/forms might exist from earlier work).
    - **Goal:** Develop user-friendly frontend interfaces for CRUD (Create, Read, Update, Delete) operations on:
        - suppliers table.
        - hpe_part_mappings table.
        - hpe_description_mappings table.
        - qb_product_mapping table.
    - **Action Items:** Design and implement React components (forms, tables/lists) and connect them to the existing backend CRUD API endpoints for these tables.
10. **Unit/Integration Tests (Backend):**
    - **Status:** PENDING.
    - **Action Item:** Add automated tests (e.g., using pytest) for critical backend logic:
        - Order processing steps in app.py.
        - HPE SKU mapping logic.
        - IIF generation logic in iif_generator.py.

        - Key functions in service modules (shipping_service.py, document_generator.py, email_service.py).
        - Database interactions.

9. Known Issues / Considerations (Updated)

- **IIF Import Sensitivities:** QuickBooks Desktop IIF import is very sensitive to file structure, character encoding (UTF-8 without BOM is generally best), and exact matching of list items (Vendors, Items, Accounts, Terms, Ship Via). The current iif_generator.py is producing files that have successfully imported in testing, but ongoing vigilance is needed if new data types or characters are introduced.
- **UPS Production Costs:** Generating labels via the production UPS API will incur real costs. Test cautiously. Understand voiding procedures if needed.
- **Postmark Deliverability:** Proper DNS setup (SPF, DKIM, DMARC, Custom Return-Path) for the sending domain is crucial for email deliverability to suppliers.
- **Scalability of Synchronous Processing:** The current process_order route is synchronous. For very high volumes, consider transitioning long-running tasks (PDF generation, external API calls like UPS) to an asynchronous task queue (e.g., Google Cloud Tasks invoking a separate Cloud Run service or Cloud Function).
- **Environment Management:** Ensure robust separation and management of environment variables and secrets for local development, staging (if any), and production.

10. Environment Setup Guide (Recap for New Developer)

- **Backend (Python/Flask):**
    1. Clone repository.
    2. Navigate to backend root (order-processing-app).
    3. Ensure Python 3.9+ is installed.
    4. Create/activate Python virtual environment.
    5. Install dependencies: pip install -r requirements.txt.
    6. Create .env file in backend root (from template, DO NOT COMMIT SECRETS). Populate with:

        - Database credentials (for local dev, often connect directly or via local Cloud SQL Proxy).
        - BigCommerce API credentials.
        - UPS API credentials (Client ID, Secret, Account #, Environment flag).
        - Postmark Server API Token, Sender/BCC addresses.
        - GCS Bucket Name (and set up local Application Default Credentials via gcloud auth application-default login for GCS/Cloud SQL access).
        - Ship From address details.
        - Other config like BC_PROCESSING_STATUS_ID, BC_SHIPPED_STATUS_ID, QUICKBOOKS_EMAIL_RECIPIENT, etc.
    7. For Local DB (Cloud SQL Proxy):
        - Install gcloud CLI and authenticate.
        - Download/run Cloud SQL Auth Proxy executable, pointing to the production Cloud SQL instance connection name (DB_CONNECTION_NAME from .env).
        - Configure .env DB variables for proxy connection (host 127.0.0.1, port 5432).
    8. Run Flask app locally: python app.py.

- **Frontend (React/Vite):**
    1. Navigate to frontend root (order-processing-app-frontend).
    2. Ensure Node.js (LTS version) and npm are installed.
    3. Install dependencies: npm install.
    4. Create .env.development in frontend root for local API proxy: VITE_API_BASE_URL=/api (or the full local backend URL).
    5. Ensure vite.config.js has proxy setup for /api pointing to local backend (e.g., http://localhost:8080 if that's where the Flask app runs).
    6. Run Vite dev server: npm run dev.

11. Deployment Process (Recap)

- **Backend (to Cloud Run):**
    1. Make code changes in order-processing-app.
    2. Update requirements.txt if new Python packages are added.
    3. Rebuild Docker image: docker build -t gcr.io/YOUR_PROJECT_ID/g1-po-app-backend-service:YOUR_TAG .
    4. Push Docker image to Artifact Registry: docker push gcr.io/YOUR_PROJECT_ID/g1-po-app-backend-service:YOUR_TAG
    5. Redeploy Cloud Run service using gcloud run deploy g1-po-app-backend-service --image gcr.io/YOUR_PROJECT_ID/g1-po-app-backend-service:YOUR_TAG --region YOUR_REGION ... (include all necessary flags for env vars, secrets, Cloud SQL instance, etc., as per your deployment script or gcloud run deploy g1-po-app-backend.txt).
- **Frontend (to Firebase Hosting):**
    1. Make code changes in order-processing-app-frontend.
    2. Update .env.production if VITE_API_BASE_URL changes.
    3. Build for production: npm run build.
    4. Deploy to Firebase: firebase deploy --only hosting.

12. Conclusion for Handover

Significant progress has been made, and the G1 PO App is now in a much more robust state. The core backend processing workflow is functional, and the critical daily IIF generation for QuickBooks Desktop has been successfully implemented and tested. Key API integrations are working, and the automated scheduling of the IIF task is in place.

The primary remaining MVP tasks include the Excel PO Export feature and comprehensive end-to-end testing of all functionalities. Following that, addressing the frontend font issue, setting up the custom domain, security hardening, and building out the UI for managing suppliers and product mappings will be key priorities. This document, along with the codebase and existing cloud infrastructure, should provide the incoming developer with a solid foundation to successfully complete the MVP and move the application towards full production readiness.