G1 PO App - Definitive Handover Report & Project Plan

Date: May 16, 2025 (Reflecting all progress and chat sessions ending May 16, 2025, CDT)

Version: 9.0 (This document supersedes all previous handover reports and serves as the single source of truth for project status and planning.)

Prepared For: Incoming Developer

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

1. Project Goal & Core Value Proposition

To develop a secure, internal web application ("G1 PO App") that automates and streamlines the purchase order (PO) and shipment process for drop-shipped items. The system ingests orders from BigCommerce, allows authorized users to manage these orders, generate necessary documentation (POs, Packing Slips, Shipping Labels), communicate with suppliers, update BigCommerce, and integrate with QuickBooks Desktop for accounting. The primary aim is to reduce manual effort, improve accuracy, and provide a centralized platform for this critical business workflow.

2. Current State of the Application (As of May 16, 2025)

The application is substantially functional with many core features implemented and refined. Significant progress has been made on handling complex order scenarios, robust authentication/authorization, integrations with external services, and UI enhancements to the main dashboard and order detail views.

Key Achievements & Functionalities:

* **Order Ingestion & Management:**

   * Successfully ingests relevant orders from BigCommerce (filtered by status, e.g., "Awaiting Fulfillment").

   * Logic includes preservation of manually set statuses and stripping of sensitive patterns from customer notes.

   * Dashboard for viewing/filtering orders (by status) and importing new orders.

   * Detailed order view (`OrderDetail.jsx`) with original order information, customer details, and line items.

   * Dynamic spare part lookup for HPE option SKUs.

   * Adaptive UI for Single vs. Multi-Supplier PO Fulfillment.

* **Dashboard UI (`Dashboard.jsx` - Major Updates):**

   * **Unified Dashboard Component:** The `Dashboard.jsx` component now handles both the "Orders" view and a "Daily Sales Report" view, determined by an `initialView` prop passed via routing from `App.jsx`.

   * **"Daily Sales Report" Integrated:**

      * Accessible via a "Daily Sales" link in the main application navigation bar (defined in `App.jsx`), which replaced the previous "Products" link.

      * Fetches data from the `/api/reports/daily-revenue` backend endpoint (`app.py`), which aggregates revenue based on **UTC days**.

      * Displays revenue for the last 14 UTC days.

      * **Styling & Layout (Daily Sales Items):**

         * Each item displays the date and its corresponding revenue total on the same line.

         * Revenue numbers are right-justified on the page/card using a CSS Grid layout for each item row.

         * Revenue numbers are styled bold and green if > $0.00, and orange if $0.00.

      * **Timezone & Data Display Logic (Daily Sales Dates):**

         * The date labels displayed in the report accurately reflect the **UTC day** for which the revenue was calculated (e.g., "May 16, 2025" represents revenue for the entirety of May 16th in UTC).

    * To prevent user confusion where "today's UTC date" might appear as "tomorrow's date" in earlier local timezones (like CDT), an entry for the **current UTC day is conditionally hidden if its sales total is $0.00**.

  * **"Products" Tab/UI Removed:** User interface elements for a "Products" tab or section have been removed from `Dashboard.jsx`. The backend API routes for product mappings (`/api/products` in `app.py`) still exist for potential future administrative use or direct data management.

* **Order Detail UI (`OrderDetail.jsx` - Recent Enhancements):**

  * **Victorious Success Message:** Upon successful order processing, a prominent "ORDER PROCESSED SUCCESSFULLY!" message is displayed in bold green font.

* **Backend Processing (`app.py` - Python/Flask):**

  * Handles order processing requests from the frontend.

  * Multi-Supplier PO Logic: Creates distinct Purchase Orders, PO Line Items, and Shipment records.

  * Generates unique, sequential PO numbers.

  * Database interaction with PostgreSQL on Google Cloud SQL.

  * The `/api/reports/daily-revenue` endpoint provides data for the Daily Sales Report, aggregating sales by UTC day.

* **Document Generation (`document_generator.py` - ReportLab):**

  * Purchase Orders (PDFs): Includes supplier info, items, costs, notes, and a "Partial fulfillment" flag. The company address in the PO header (`COMPANY_ADDRESS_PO_HEADER`) is now set to an empty string.

  * Packing Slips (PDFs):

    * Differentiates items in the current shipment vs. items shipping separately.

    * **Customer Phone Number Removed:** The customer's phone number is no longer included in the "Ship To" address block.

    * **Item Descriptions:** Logic updated in `app.py` to ensure item descriptions on packing slips use the "pure" value from `hpe_description_mappings.po_description` if available, or the original BigCommerce item name as a fallback, without suffixes like " - clean pull".

* **Shipping Label Generation (`shipping_service.py` - UPS API):**

  * Generates UPS shipping labels (PDF via GIF conversion) using OAuth 2.0.

  * Handles mapping of shipping method names to UPS service codes.

* **Email Communication (`email_service.py` - Postmark API):**

  * Emails PO PDF, Packing Slip PDF, and UPS Label PDF (if generated) to suppliers.

  * Sends daily IIF batch emails.

* **Cloud Storage (`app.py` - Google Cloud Storage):**

  * Uploads generated POs, Packing Slips, and Labels to GCS.

* **BigCommerce Integration (`shipping_service.py`, `app.py`):**

  * Retrieves order data.

  * Refined logic for shipment creation and final order status updates.

* **QuickBooks Integration (`iif_generator.py`, `app.py`):**

  * Generates and emails a daily IIF file for Purchase Orders.

  * Automated triggering via Google Cloud Scheduler.

* **Authentication & Authorization (Firebase):**

  * Robust user authentication via Firebase (Google Sign-In, project `g1-po-app-77790`).

  * Authorization using Firebase Custom Claims (`isApproved: true`).

  * Backend API routes are protected using a `@verify_firebase_token` decorator.

  * **Firebase API Key Issue Resolved:** Ensured correct Firebase configuration and API key are used in the frontend.

  * **GCS Signed URL IAM Permission:** The necessity of granting "Service Account Token Creator" role to the Cloud Run service account for signed URL generation has been identified and instructed.

* **Routing (`App.jsx` - Major Updates):**

  * The main navigation bar now links "Daily Sales" to the `/dashboard/sales` route.

  * The `/dashboard` route (and `/`) renders the `Dashboard` component with `initialView="orders"`.

  * The `/dashboard/sales` route renders the `Dashboard` component with `initialView="dailySales"`.

  * Routes and component imports related to the previous "Products" page have been removed.

* **Error Handling and Logging:** Implemented at various levels.

3. Technology Stack & Project Layout

* **Backend (`order-processing-app` directory):**

  * Python 3.9+, Flask, SQLAlchemy, pg8000, Gunicorn.

  * Key files: `app.py`, `shipping_service.py`, `email_service.py`, `document_generator.py`, `iif_generator.py`.

  * Dependencies: `requirements.txt`.

  * Configuration: `.env` file for local, Google Cloud Secret Manager for deployed.

* **Frontend (`order-processing-app-frontend` directory):**

  * React (Vite), `react-router-dom`, Firebase Client SDK, `fetch` API.

   * Key components: `App.jsx`, `Dashboard.jsx`, `OrderDetail.jsx`, `Login.jsx`, `AuthContext.jsx`, `ProtectedRoute.jsx`, `SupplierList.jsx`, etc.

   * Configuration: `.env.development`, `.env.production` for `VITE_API_BASE_URL` and Firebase SDK config.

* **Database:** PostgreSQL on Google Cloud SQL.

* **Cloud Platform (GCP):**

   * Cloud Run (backend).

   * Cloud SQL (database).

   * Artifact Registry (Docker images).

   * Secret Manager (secrets).

   * Cloud Storage (GCS for documents).

   * Cloud Scheduler (IIF task).

* **Firebase:**

   * Firebase Hosting (frontend).

   * Firebase Authentication (user auth for project `g1-po-app-77790`).

* **External APIs:**

   * UPS API (shipping labels).

   * BigCommerce API (orders).

   * Postmark API (emails).

4. Key Code Files & Their Roles (Highlighting Recent Changes)

* **`app.py` (Backend Flask application - Canvas ID: `app_py_signed_urls_complete_v2`):**

   * Provides all API endpoints.

   * `/api/orders/<order_id>/process`:

      * Handles single and multi-supplier PO generation.

    * **Recent Change:** Removed the call to `email_service.send_quickbooks_data_email`.

  * `/api/reports/daily-revenue`: Aggregates sales data by UTC day for the Daily Sales Report.

  * Contains logic for order ingestion, database interactions, and calls to service modules.

* **`OrderDetail.jsx` (Frontend - Canvas ID: `OrderDetail_jsx_final_touches_v2`):**

  * Displays details for a single order and handles the PO processing form.

  * **Recent Changes:**

    * Displays "ORDER PROCESSED SUCCESSFULLY!" upon success.

    * Formats PO information as "PO # \[Number] sent to \[Supplier Name]".

* **`Dashboard.jsx` (Frontend):**

  * Displays the main order list and the new "Daily Sales Report".

  * Conditionally renders views based on the `initialView` prop.

  * **Daily Sales Report:** Displays UTC-dated revenue, hides today's UTC date if sales are zero, styles revenue numbers.

* **`App.jsx` (Frontend):**

  * Manages top-level application routing.

  * **Recent Change:** Navigation bar updated to link "Daily Sales" to `/dashboard/sales`. Product-related routes removed.

* **`document_generator.py` (Backend):**

  * Generates PO and Packing Slip PDFs using ReportLab.

  * **Recent Changes:**

    * `COMPANY_ADDRESS_PO_HEADER` global variable set to `""`.

    * Customer phone number removed from the "Ship To" address on packing slips.

    * Relies on `app.py` to pass "pure" item descriptions (from `hpe_description_mappings` or original BigCommerce name) for packing slip items.

* **`email_service.py` (Backend):**

  * Handles sending emails via Postmark. The `send_quickbooks_data_email` function still exists but is no longer called by `app.py`.

* **Other Service Modules (Backend):** `shipping_service.py`, `iif_generator.py`.

* **Frontend Contexts/Components:** `AuthContext.jsx`, `ProtectedRoute.jsx`, etc.

5. Developer Setup & Environment

* **Backend (`order-processing-app` directory):**

  * Python 3.9+.

  * Virtual environment (e.g., `python -m venv venv`).

  * Install dependencies: `pip install -r requirements.txt`.

  * Local `.env` file for secrets (DB credentials, API keys, GCS bucket name, Firebase Project ID, etc.).

  * `GOOGLE_APPLICATION_CREDENTIALS` environment variable pointing to a Firebase Admin SDK service account key JSON file for local development. This service account key *must* belong to the Firebase project (`g1-po-app-77790`) and have necessary permissions, including the ability to act as itself to sign tokens if you test signed URL generation locally with this key (less common for `generate_signed_url` which typically uses the runtime identity on GCP).

  * Google Cloud SQL Auth Proxy for connecting to the Cloud SQL PostgreSQL database locally.

* **Frontend (`order-processing-app-frontend` directory):**

  * Node.js (LTS version recommended).

  * Install dependencies: `npm install` (or `yarn install`).

  * `src/contexts/AuthContext.js` (or `src/firebase.js`) must contain the correct Firebase project configuration snippet from the Firebase console (API key, auth domain, etc.). This can be hardcoded or, preferably, loaded from Vite environment variables.

  * `.env.development` and/or `.env.local` file(s) in the frontend project root for Vite:

    * `VITE_API_BASE_URL` (e.g., `http://127.0.0.1:8080/api` for local backend).

* `VITE_FIREBASE_API_KEY`, `VITE_FIREBASE_AUTH_DOMAIN`, `VITE_FIREBASE_PROJECT_ID`, etc.

* **Running Locally:**

* Start backend: `flask run --port=8080` (or `python app.py`).

* Start frontend: `npm run dev`.

* **Deployment:**

* Backend: Dockerized, pushed to Google Artifact Registry, deployed to Google Cloud Run. **Crucial:** The Cloud Run service account needs the "Service Account Token Creator" role on itself for GCS Signed URLs, and appropriate permissions for Cloud SQL, GCS (read/write to bucket), Secret Manager, etc.

* Frontend: Built with `npm run build`, deployed to Firebase Hosting.

6. Detailed Plan for Completion & Future Tasks

Immediate Next Steps (Post-Handover):

1. **Task: Verify Backend Signed URL Generation and Frontend Consumption.**

* **Action:**

* Confirm the "Service Account Token Creator" IAM role is active for the Cloud Run service account.

* Deploy the latest `app.py` (from Canvas `app_py_signed_urls_complete_v2`).

* Deploy the latest `OrderDetail.jsx` (from Canvas `OrderDetail_jsx_final_touches_v2`).

* Process an order.

* The text "PO # \[Number] sent to \[Supplier Name]" should display correctly.

* **File(s) Involved:** `app.py`, `OrderDetail.jsx`, GCP IAM.

2. **Task: Verify Customer-Facing SKU/Description on Packing Slip (Phase 1 from PDF).**

* **Details:** In `app.py`'s `process_order` loop, the logic for `prepared_items_in_this_shipment` and `prepared_items_shipping_separately` now attempts to use `hpe_description_mappings.po_description` or falls back to `line_item_name`.

* **Action:** Thoroughly test this with various items (mapped, unmapped, complex SKUs) to ensure the descriptions on the packing slip are always the desired "pure" customer-facing ones, without suffixes like " - clean pull" unless that's part of the actual mapped description.

* **File(s) Involved:** `app.py`, `document_generator.py` (to confirm it uses the passed `name` field).

3. **Task: Comprehensive End-to-End Testing of Single and Multi-Supplier Workflows (Phase 1 from PDF).**

* **Action:** Critical after recent changes. Test all aspects: UI, backend, document generation (PO, Packing Slip content verification), label generation, emails, GCS uploads (confirm signed URLs work), BigCommerce updates (shipment creation, final status, single customer email), local DB updates. Test edge cases.

* **File(s) Involved:** Full application stack.

Phase 2: MVP Task Completion

4. **Task: PO Export Feature (Excel - Backend & Frontend).**

* **Backend (`app.py`):** Implement `GET /api/exports/pos` using `openpyxl`.

* **Frontend (`Dashboard.jsx`):** Add UI to trigger export.

5. **Task: Finalize Authentication & Authorization Robustness.**

* Frontend error handling for 401/403.

* Documentation for `setAdminClaim.cjs` (or prioritize Admin UI).

* Re-verify Cloud Run IAM permissions.

Phase 3: UI/UX Enhancements & Remaining Fixes

6. **Task: Supplier and Product/Mapping Management UI.**

   * Implement React components for CRUD on `suppliers`.

   * Re-evaluate need/priority for UI for `hpe_part_mappings`, `hpe_description_mappings`, `qb_product_mapping`, `products`. If needed, design entry point (e.g., Admin section).

7. **Task: Frontend - Eloquia Font Issue.**

   * Resolve font rendering on deployed site.

8. **Task: Loading Indicators and UX Refinements.**

   * Audit API-calling components for consistent loading indicators.

Phase 4: Infrastructure & Deployment Finalization

9. **Task: Postmark - External Domain Sending Approval.**

   * Ensure Postmark is fully configured (DKIM, SPF, Custom Return-Path).

10. **Task: Cloudflare DNS for Production Domains.**

   * Configure `g1po.com` (frontend) and `api.g1po.com` (backend).

Phase 5: Long-Term Improvements & Best Practices

11. **Task: Backend Daily Revenue Aggregation Timezone.**

   * **Details:** If business reporting requires alignment with a specific local timezone (e.g., CDT) for the Daily Sales Report, refactor the backend query in `app.py` (`/api/reports/daily-revenue`) to group revenue by that target timezone's day boundaries.

12. **Task: Security Hardening.**

   * Cloud Run Ingress (consider IAP).

   * Comprehensive server-side input validation.

   * Structured, detailed, leveled logging.

13. **Task: Asynchronous Task Processing.**

   * For `/api/orders/<id>/process`, consider Google Cloud Tasks.

14. **Task: Admin User Approval UI.**

   * Implement an admin interface for managing `isApproved` claims.

15. **Task: Data Integrity for QuickBooks Lists (TERMS and SHIPVIA).**

   * Ensure data matches QuickBooks lists for IIF generation.

16. **Task: Automated Unit/Integration Tests.**

   * Add Pytest (backend) and Jest/React Testing Library (frontend) tests.

7. Known Issues & Considerations

* **IIF Import Sensitivities:** QuickBooks Desktop IIF import is sensitive.

* **UPS Production Costs:** Live UPS API will incur costs.

* **Postmark Deliverability:** DNS setup is critical.

* **Scalability of Synchronous Processing:** Current order processing is synchronous.

* **Environment Management:** Maintain strict separation for dev/prod.

* **Custom Claim Management:** `setAdminClaim.cjs` is manual.

* **Firebase Quotas:** Monitor usage.

* **Service Account Key Security:** Protect any downloaded service account keys.

* **Timezone for Reporting:** Daily Sales Report uses UTC day aggregation. Consider backend changes for local business day alignment if required.

8. Conclusion for Handover

The G1 PO App is a powerful tool with significant functionality already in place. Subsequent phases will further enhance the application's capabilities and robustness. This document provides a comprehensive snapshot and a clear path forward.