G1 PO App - Definitive Handover Report & Project Plan

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

Version: 8.0 (This document supersedes all previous handover reports, including v7.0, 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.

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**. This avoids showing a future-looking date with no activity.

    * **"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.

* **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.

    * Packing Slips (PDFs): Differentiates items in the current shipment vs. items shipping separately.

* **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 to suppliers.
    * Sends structured PO data (for QuickBooks) to an internal email.
    * 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 (creating individual shipments without changing overall order status) and final order status updates (setting to "Shipped" only when all items are covered, ensuring a single customer notification).

* **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 protected using a `@verify_firebase_token` decorator.
    * **Firebase API Key Issue Resolved:** Correct Firebase configuration (API key, etc.) is implemented in the frontend, resolving previous `auth/invalid-api-key` errors.

* **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 throughout the application.

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 for HTTP requests.

  * 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

* **`App.jsx`:**
  * Manages top-level application routing using `react-router-dom`.
  * Contains the main navigation bar. "Products" link changed to "Daily Sales", linking to `/dashboard/sales`.
  * Defines routes for `/dashboard` (shows orders) and `/dashboard/sales` (shows sales report), both rendering the `Dashboard` component with different `initialView` props. Product-related routes are removed.
* **`Dashboard.jsx`:**
  * A versatile component displaying either "Orders" or "Daily Sales Report" based on the `initialView` prop.
  * Manages state for orders data, status counts, ingestion, daily revenue, loading, and errors.
  * Conditionally renders UI based on `currentView` state.

  * Handles data fetching for both views.

  * **Daily Sales Display Logic:**

    * Formats dates to display as UTC dates (e.g., "May 16, 2025").

    * Hides the entry for the current UTC day if its revenue is $0.00.

    * Styles revenue numbers (bold/green for positive, orange for zero).

    * Uses CSS Grid (via `Dashboard.css`) for layout, ensuring revenue numbers are right-justified.

* **`app.py` (Backend Flask application):**

  * Provides all API endpoints.

  * `/api/reports/daily-revenue` endpoint aggregates sales data by UTC days and serves the Daily Sales Report.

  * Contains logic for order ingestion, processing, document generation, and integrations.

* **`Dashboard.css`:**

  * Contains styling for the Dashboard component.

  * Includes styles for `.daily-revenue-list` and `.daily-revenue-item` using CSS Grid for the Daily Sales report layout.

* **`AuthContext.jsx` (or `contexts/AuthContext.js`):**

  * Manages Firebase authentication state.

  * Contains Firebase SDK initialization. **Crucial:** Must have the correct Firebase project configuration (API key, auth domain, project ID, etc.).

* **Service Modules (Python backend):**

  * `document_generator.py`: PDF generation.

  * `shipping_service.py`: UPS label and BigCommerce shipment/status updates.

  * `email_service.py`: Email sending via Postmark.

  * `iif_generator.py`: QuickBooks IIF file generation.

* **Other Frontend Components:** `OrderDetail.jsx`, `Login.jsx`, `SupplierList.jsx`, `EditSupplierForm.jsx`, `SupplierForm.jsx`, `ProtectedRoute.jsx` etc.

5. Developer Setup & Environment

* **Backend:**
  * 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, BigCommerce tokens, etc.).
  * `GOOGLE_APPLICATION_CREDENTIALS` environment variable pointing to Firebase Admin SDK service account key JSON for local development.
  * Google Cloud SQL Auth Proxy for local database connection.
* **Frontend:**
  * 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 from the Firebase console (can be direct or via 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`).
    * `VITE_FIREBASE_API_KEY`, `VITE_FIREBASE_AUTH_DOMAIN`, `VITE_FIREBASE_PROJECT_ID`, etc.
* **Running Locally:**
  * Start backend: `flask run` or `python app.py`.
  * Start frontend: `npm run dev` or `yarn dev`.
* **Deployment:**
  * Backend: Dockerized, pushed to Google Artifact Registry, deployed to Google Cloud Run.

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

* Ensure Cloud Run service accounts have necessary IAM permissions (Cloud SQL, GCS, Secret Manager, Firebase Admin SDK).

6. Detailed Plan for Completion & Future Tasks

Phase 1: Stabilize & Complete Core Multi-Supplier Fulfillment (Immediate Priority)

1. **Task:** Verify Customer-Facing SKU/Description on Packing Slip for "IN THIS SHIPMENT" items.

* **Details:** In `app.py`'s `process_order` loop, ensure `prepared_items_in_this_shipment` uses customer-facing SKU/description, mapping back from internal/supplier SKUs if necessary, consistent with the "SHIPPING SEPARATELY" section logic.

* **File(s) Involved:** `app.py`.

2. **Task:** Comprehensive End-to-End Testing of Single and Multi-Supplier Workflows.

* **Details:** Test UI interactions, backend processing, document generation (POs, Packing Slips - verify "Partial fulfillment", "IN THIS SHIPMENT", "SHIPPING SEPARATELY" sections), UPS Label generation, email content and attachments, GCS uploads, BigCommerce shipment creation and order status updates (ensure single final "shipped" email to customer), and local database updates. Include edge case testing.

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

Phase 2: Finalize Dashboard and Core UX

3. **Task:** Thorough End-to-End Testing of Daily Sales Report.

* **Details:** Verify data accuracy against source, correct UTC date display, conditional hiding of current UTC day if revenue is zero, styling (right-justification of numbers, color coding), and navigation to/from the report via the top bar.

* **File(s) Involved:** `Dashboard.jsx`, `App.jsx`, `app.py` (data source).

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

* **Backend (`app.py`):** Implement `GET /api/exports/pos` endpoint using `openpyxl` to generate an `.xlsx` file of PO details (joined from `purchase_orders`, `po_line_items`, `suppliers`). Allow basic filtering (date range, supplier).

* **Frontend (`Dashboard.jsx` or new component):** Add UI elements (button, date pickers, supplier dropdown) to trigger this export and handle the file download.

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

* Thorough testing of login, logout, session persistence, and protected routes.

* Implement robust frontend error handling for 401/403 API responses (display clear messages, consider auto-logout to login page).

* Provide clear documentation for the administrator on using `setAdminClaim.cjs` securely (or prioritize Admin UI in Phase 5).

* Re-verify Cloud Run service account IAM permissions, adhering to the principle of least privilege.

Phase 3: UI/UX Enhancements & Remaining Fixes

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

* **Details:** Design and implement user-friendly React components for CRUD operations on `suppliers`. Consider if CRUD UI for `hpe_part_mappings`, `hpe_description_mappings`, `qb_product_mapping`, and `products` tables is still a priority given the "Products" main navigation was removed. If so, this UI would need a new entry point (e.g., under an "Admin Settings" section).

* Connect to existing backend CRUD APIs or create/enhance them as needed.

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

* **Details:** Investigate and resolve any custom font rendering issues on the deployed Firebase site. Ensure font files are correctly referenced, licensed, and served.

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

* **Details:** Audit all components that make API calls to ensure consistent, clear, and non-intrusive loading state indicators are used. Improve user feedback for background operations or delays.

Phase 4: Infrastructure & Deployment Finalization

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

* **Details:** Ensure the Postmark account is fully approved and configured (DKIM, SPF, Custom Return-Path) for sending emails from the application's designated domain to external supplier domains to maximize deliverability. Monitor bounce rates and spam complaints.

10. **Task:** Cloudflare DNS for `g1po.com` & `api.g1po.com`.

* **Frontend:** Add `g1po.com` (or chosen production domain) as a custom domain in Firebase Hosting and update DNS records in Cloudflare.

* **Backend (Recommended):** Set up `api.g1po.com` (or chosen API domain) pointing to the Cloud Run service, map this custom domain in Cloud Run settings, and update `VITE_API_BASE_URL` in the frontend production environment configuration.

Phase 5: Long-Term Improvements & Best Practices

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

* **Details:** Currently, the `/api/reports/daily-revenue` endpoint in `app.py` aggregates sales based on UTC days. If business reporting requires alignment with a specific local operational timezone (e.g., CDT/CST), refactor the backend SQL query to group revenue by that target timezone's day boundaries. This will make the "Daily Sales Report" data align with local business days.

* **File(s) Involved:** `app.py`.

12. **Task:** Security Hardening.

* **Cloud Run Ingress:** Re-evaluate if GCP-level authentication (e.g., IAP with OIDC) should be layered on top of the current application-level Firebase token verification for the main backend service, especially if the API endpoint is made public.

* **Input Validation:** Conduct a comprehensive review of all backend API endpoints for robust server-side input validation on all incoming data (query parameters, JSON bodies) to prevent common vulnerabilities (SQLi, XSS through stored data, etc.).

* **Logging:** Implement more structured, detailed, and leveled logging on the backend (Python's `logging` module, effective use of Google Cloud Logging) for better monitoring, auditing, and troubleshooting.

13. **Task:** Asynchronous Task Processing for Intensive Operations.

* **Details:** For potentially long-running operations within the `/api/orders/<id>/process` endpoint (especially if batch sizes grow or document generation/API calls become more complex), refactor to use asynchronous task queues like Google Cloud Tasks. This will improve API responsiveness and scalability.

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

* **Details:** Design and implement a simple, secure admin interface within the G1 PO App itself (e.g., a new route like `/admin/users`) for an administrator (identified by a specific Firebase custom claim like `isAdmin: true`) to manage `isApproved` claims for other users. This would replace the manual `setAdminClaim.cjs` script and improve usability.

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

* **Details:** If more complete IIF files are desired, implement functionality to ensure TERMS and SHIPVIA data used in POs matches QuickBooks lists exactly. This may involve UI changes for data entry (dropdowns populated from a managed list) or more complex backend mapping.

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

* **Details:** Incrementally add automated tests (e.g., Pytest for backend API endpoints and service logic; Jest/React Testing Library for critical frontend components and user flows) to improve code quality, ensure regressions are caught early, and facilitate safer refactoring.

7. Known Issues & Considerations

* **IIF Import Sensitivities:** QuickBooks Desktop IIF import can be finicky. Thorough testing of generated IIF files with the target QuickBooks version is crucial.

* **UPS Production Costs:** Using the production UPS API will incur real shipping costs. Ensure proper testing in a UPS sandbox environment if available before full production use with live credentials.

* **Postmark Deliverability:** Correct DNS setup (DKIM, SPF, Custom Return-Path) for the sending domain is critical for good email deliverability to suppliers.

* **Scalability of Synchronous Processing:** The current order processing endpoint (`/api/orders/<id>/process`) is synchronous. For very high volumes or many complex operations per order, this could lead to timeouts or slow responses. (See Phase 5, Task 13).

* **Environment Management:** Strict separation and correct configuration for local development, staging (if implemented), and production environments are essential. `VITE_API_BASE_URL` and backend `.env`/Secret Manager credentials must be accurate for each environment.

* **Custom Claim Management:** The `setAdminClaim.cjs` script for setting user approval is manual. A UI is a future enhancement (Phase 5, Task 14).

* **Firebase Quotas:** Monitor Firebase Authentication and other Firebase service usage against project quotas.

* **Service Account Key Security:** The Firebase Admin SDK service account key JSON file is highly sensitive. **Never commit it to Git.** Store it securely and use environment variables or secure local storage methods for local development. For deployed environments (Cloud Run), rely on the service's identity and IAM roles.

* **Timezone Handling for Reporting:** The Daily Sales Report currently displays data aggregated by UTC days, with UTC date labels. If the business requires reports based on local operational days (e.g., sales from midnight to midnight CDT), the backend data aggregation logic needs to be modified (Phase 5, Task 11).

8. Conclusion for Handover

The G1 PO App is in a robust state, with core functionalities for order processing and dashboard reporting well-established. The recent dashboard enhancements, including the

Daily Sales Report, provide valuable insights. The immediate priorities for the incoming developer should be the thorough testing and stabilization of the multi-supplier fulfillment workflow (Phase 1 tasks). Following this, completing the remaining MVP features like PO export and further UI/UX refinements will significantly enhance the application's utility.

This comprehensive report aims to provide all necessary context, current status, and a clear roadmap for the continued development and success of the G1 PO App.