G1 PO App - Full Handover Report & Project Plan

Date: May 15, 2025 (Reflecting progress and chat session ending May 15, 2025, CDT) **Version:** 6.0 (Supersedes all previous handover reports and incorporates all development to date) **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 15, 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, and integrations with external services.

Key Achievements & Functionalities:

- **Order Ingestion:** 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. ([Source Handover v4.0: 9, 29, 30, 74, 75, 132, 133])
- **User Interface (React/Vite Frontend):**
  - Dashboard for viewing/filtering orders and importing new orders. ([Source Handover v4.0: 17, 49, 90])
  - Detailed order view (OrderDetail.jsx) with:
    - Display of original order information, including customer details and line items from BigCommerce.
    - Dynamic spare part lookup and display for HPE option SKUs, with links to Brokerbin. ([Source Handover v4.0: 21, 55, 56, 76-78, 93-99, 134, 135, 151-154, 193])
    - **Adaptive UI for Single vs. Multi-Supplier PO Fulfillment:**

            - Users can select a strategy to process the order via a single supplier or multiple suppliers.
            - **Single-Supplier Mode:** Allows editing of purchase items (SKU, description, quantity, unit cost), PO notes, and global shipment method/weight.
            - **Multi-Supplier Mode:**
                - Original line items are listed for assignment to different suppliers.
                - Dynamic "PO Draft" sections appear for each unique assigned supplier.
                - Each draft section allows input for:
                    - Specific PO Notes for that supplier.
                    - Unit cost for each original line item assigned to that supplier.
                    - Shipment Method (dropdown defaulting to original customer selection, includes "UPS Next Day Air Early A.M.").
                    - Shipment Weight (lbs).
        - Click-to-copy functionality for Order #.
        - Buttons for processing orders and manual status updates, with consistent global styling. ([Source Handover v4.0: 6, 48, 50-52, 100-102, 133])
    - Browser page title and favicon are set. ([Source Handover v4.0: 7, 44, 45, 85, 86])
- **Backend Processing (app.py - Python/Flask):**
    - Handles order processing requests from the frontend.
    - **Multi-Supplier PO Logic:**
        - Accepts an assignments array payload, where each object defines a PO (supplier, items, notes, costs, shipment details).

        - Loops through assignments to create distinct Purchase Orders, PO Line Items, and Shipment records in the database.
        - Generates unique, sequential PO numbers (SQL logic for MAX(CAST(po_number AS INTEGER)) from a subquery with ` CAST(po_number AS TEXT) ~ '^[0-9]+$' is implemented).
    - Database interaction with PostgreSQL on Google Cloud SQL. ([Source Handover v4.0: 27, 28, 62, 63])
    - Integration with custom service modules for specific tasks.
- **Document Generation (document_generator.py - ReportLab):**
    - **Purchase Orders (PDFs):** Generated with supplier info, items, costs, notes. Now includes an is_partial_fulfillment flag to state "Partial fulfillment of G1 Order #" when applicable.
    - **Packing Slips (PDFs):** Logic and PDF rendering **updated** to:
        - Accept items_in_this_shipment and items_shipping_separately.
        - Display an "IN THIS SHIPMENT" section with its items (standard black font).
        - Display a "SHIPPING SEPARATELY" section (if applicable) with its items (grey font).
        - Include a reference PO number (po_number_for_slip).
- **Shipping Label Generation (shipping_service.py - UPS API):**
    - Generates UPS shipping labels (PDF via GIF conversion) using OAuth 2.0. ([Source Handover v4.0: 12])
    - Handles mapping of frontend shipping method names to UPS service codes (including "UPS Next Day Air Early A.M." to code "14").
    - **Now successfully generates labels for individual POs in a multi-supplier scenario if shipment weight and method are provided for that PO.**
- **Email Communication (email_service.py - Postmark API):**
    - Emails PO PDF, Packing Slip PDF, and UPS Label PDF (if generated) to the selected supplier for each PO. ([Source Handover v4.0: 13])

    - Sends structured PO data (for QuickBooks import preparation) to a designated internal email address for each PO.
    - Sends daily IIF batch emails.
- **Cloud Storage (app.py - Google Cloud Storage):**
    - Uploads generated POs, Packing Slips, and Labels to GCS. ([Source Handover v4.0: 14])
- **BigCommerce Integration (shipping_service.py, app.py):**
    - Retrieves order data.
    - **Shipment Creation & Status Updates (Refined Logic):**
        - shipping_service.py has been refactored to provide:
            - create_bigcommerce_shipment(): Creates a shipment record in BC with specific items and tracking, *without* changing the overall order status.
            - set_bigcommerce_order_status(): Updates the overall BC order status to a given status ID.
        - app.py's process_order function now:
            - Calls create_bigcommerce_shipment() for each PO that has a tracking number.
            - After processing all POs in a batch, it determines if all original order items have been covered.
            - If all items are covered, it calls set_bigcommerce_order_status() to update the BC order to the final "Shipped" status (e.g., using BC_SHIPPED_STATUS_ID). This ensures the customer receives only one "Order Shipped" email from BigCommerce when the entire order is fulfilled.
- **QuickBooks Integration (iif_generator.py, app.py):**
    - Generates a daily IIF (Intuit Interchange Format) file for Purchase Orders processed on the previous day.
    - IIF file is emailed to a designated address.

    - Automated triggering via Google Cloud Scheduler is set up and functional. ([Source Handover v4.0: 3, 18, 35, 36, 66, 67, 127, 128])
- **Authentication & Authorization (Firebase):**
    - Robust user authentication via Firebase (Google Sign-In, project g1-po-app-77790). ([Source Handover v4.0: 5, 19, 37, 38, 65, 71, 73, 83, 84, 87, 110-112, 113-125, 129-131, 187-190, 192, 200, 201, 207, 213])
    - Authorization using Firebase Custom Claims (isApproved: true).
    - Backend API routes are protected using a @verify_firebase_token decorator.
- **Error Handling and Logging:** Implemented at various levels, with ongoing potential for enhancement. Debug messages have been instrumental in troubleshooting.

3.1. Known Issues/Considerations from Handover Report v4.0 (Section 10) - Still Relevant:

- **IIF Import Sensitivities:** QuickBooks Desktop IIF import remains sensitive.
- **UPS Production Costs:** Real costs will be incurred with production UPS API.
- **Postmark Deliverability:** DNS setup (DKIM, SPF, Return-Path) for the sending domain is crucial for good deliverability. ([Source Handover v4.0: 166])
- **Scalability of Synchronous Processing:** The current /process endpoint is synchronous. For very high volumes or extremely long operations within it (e.g., many complex documents for many POs), consider asynchronous task queues (e.g., Cloud Tasks) in the future. ([Source Handover v4.0: 183])
- **Environment Management:** Maintain robust separation and configuration for local dev, staging (if any), and production. VITE_API_BASE_URL and backend .env/Secret Manager credentials need to be correct per environment. ([Source Handover v4.0: 185, 186])
- **Custom Claim Management:** Current setAdminClaim.cjs is manual. A UI for this is a future enhancement. ([Source Handover v4.0: 146, 147, 188])
- **Firebase Quotas:** Monitor Firebase Authentication quotas. Current "auth/quota-exceeded" issues have been resolved, but efficient API usage is important. ([Source Handover v4.0: 189, 190])

- **Service Account Key Security:** The Firebase Admin SDK service account key (for setAdminClaim.cjs and local backend dev) is highly sensitive. Never commit to Git. ([Source Handover v4.0: 191, 192])

4. Project Layout & Technology Stack (Recap from Handover Report v4.0, Section 4 & 5)

- **Backend (order-processing-app directory):**
    - Python 3.9+, Flask, SQLAlchemy, pg8000, Gunicorn.
    - Key files: app.py (main application), 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, axios (or Workspace API), Firebase Client SDK.
    - Key components: App.jsx, Dashboard.jsx, OrderDetail.jsx, Login.jsx, AuthContext.jsx, ProtectedRoute.jsx.
    - Configuration: .env.development, .env.production for VITE_API_BASE_URL.
- **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).
- **APIs:** UPS (shipping labels), BigCommerce (orders), Postmark (emails).

5. Detailed Plan for Completion & Future Tasks

This plan integrates remaining MVP tasks from "Handover Report v4.0 (Section 9)" with new items and priorities based on recent progress.

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, when preparing prepared_items_in_this_shipment, the SKU/description currently comes from po_item_detail (which is derived from the frontend payload's po_line_items for that assignment).
- **Action:** Review if this is always the correct customer-facing representation. If the PO uses internal/supplier SKUs, ensure these are mapped back to the original customer-facing SKU/description (e.g., using original_order_line_item_id to reference local_order_line_items_list and potentially applying get_hpe_mapping_with_fallback) for the packing slip section "IN THIS SHIPMENT". The "SHIPPING SEPARATELY" section already does this.
- **File(s) Involved:** app.py.

2. **Task: Comprehensive End-to-End Testing of Single and Multi-Supplier Workflows.**
    - **Details:** This is now the most critical step for the recently developed features.
    - **Action:**
        - Test order ingestion.
        - Thoroughly test the **single-supplier PO processing flow**:
            - UI interaction in OrderDetail.jsx (editing items, notes, shipment details).
            - Backend processing in app.py.
            - Generation and content of PO PDF (should say "Fulfillment of...").
            - Generation and content of Packing Slip PDF (should list all items under "IN THIS SHIPMENT", "SHIPPING SEPARATELY" section should be absent or empty).
            - UPS Label generation.
            - Email to supplier (with all 3 attachments).
            - QB data email.
            - GCS uploads.

    - BigCommerce shipment creation and order status update to final "Shipped" status (verify single customer email).
    - Local DB updates.
- Thoroughly test the **multi-supplier PO processing flow**:
    - UI interaction in OrderDetail.jsx (assigning items to different suppliers, per-supplier PO notes, per-item unit costs, per-PO shipment method/weight).
    - Backend processing in app.py.
    - Generation and content of individual PO PDFs (should say "Partial fulfillment of..." if >1 PO).
    - Generation and content of individual Packing Slip PDFs (verify "IN THIS SHIPMENT" for that PO's items in black, and "SHIPPING SEPARATELY" for other original order items in grey).
    - UPS Label generation for each PO where shipment details were provided.
    - Individual emails to each supplier with their specific PO, packing slip, and label (if generated).
    - Individual QB data emails for each PO.
    - GCS uploads for all documents per PO.
    - Individual BigCommerce shipment creation for each PO with a tracking number.
    - **Verify BigCommerce order status:** It should only change to the final "Shipped" status after the *last* PO that completes the original order is processed. Intermediate shipments should ideally result in "Partially Shipped" (often handled by BC automatically) or not change the main status until the end. Verify customer only gets one final "shipped" email.
    - Local DB updates.
- Test edge cases: orders with no line items, items with zero cost, invalid inputs.
- Test all authentication and authorization paths.

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

Phase 2: MVP Task Completion (from Handover Report v4.0)

3. **Task: PO Export Feature (Excel - Backend & Frontend).** ([Source Handover v4.0: 155-159])
   - **Backend (app.py):** Implement GET /api/exports/pos endpoint using openpyxl to generate and return an .xlsx file of PO details (joined from purchase_orders, po_line_items, suppliers). Allow basic filtering (date range, supplier).
   - **Frontend (Dashboard.jsx):** Add UI (button, filters) to trigger this export and handle the file download.
4. **Task: Finalize Authentication & Authorization Robustness.** ([Source Handover v4.0: 139-149])
   - Thorough testing of login, logout, session persistence (largely done).
   - Robust frontend error handling for 401/403 errors from API calls (display clear messages, consider auto-logout). ([Source Handover v4.0: 144, 145])
   - Provide clear documentation for the administrator on using setAdminClaim.cjs securely. ([Source Handover v4.0: 146])
   - Re-verify Cloud Run service account IAM permissions (principle of least privilege).

Phase 3: UI/UX Enhancements & Remaining Fixes

5. **Task: Supplier and Product/Mapping Management UI.** ([Source Handover v4.0: 177-179])
   - Design and implement user-friendly React components for CRUD operations on suppliers, hpe_part_mappings, hpe_description_mappings, qb_product_mapping, and products tables.
   - Connect to existing (or create if needed) protected backend CRUD APIs for these.
6. **Task: Frontend - Eloquia Font Issue.** ([Source Handover v4.0: 61, 167])
   - Investigate and resolve font rendering issue on the deployed Firebase site. Ensure font files are correctly referenced and served.

7. **Task: Loading Indicators and UX Refinements.** ([Source Handover v4.0: 154])
    - Audit all API-calling components for consistent and clear loading state indicators.
    - Improve user feedback for background operations.

Phase 4: Infrastructure & Deployment Finalization

8. **Task: Postmark - External Domain Sending Approval.** ([Source Handover v4.0: 166])
    - Ensure the Postmark account is fully approved and configured (DKIM, SPF, Custom Return-Path) for sending emails from your desired domain to external supplier domains to maximize deliverability.
9. **Task: Cloudflare DNS for g1po.com & api.g1po.com.** ([Source Handover v4.0: 168-170])
    - Frontend: Add g1po.com as a custom domain in Firebase Hosting and update DNS records in Cloudflare.
    - Backend (Recommended): Set up api.g1po.com pointing to the Cloud Run service, map this custom domain in Cloud Run, and update VITE_API_BASE_URL in the frontend production environment configuration.

Phase 5: Long-Term Improvements & Best Practices

10. **Task: Security Hardening.** ([Source Handover v4.0: 171-176])
    - **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.
    - **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.
    - **Logging:** Implement more structured, detailed, and leveled logging on the backend (Python's logging module, effective use of Google Cloud Logging) for better monitoring and troubleshooting.
11. **Task: Asynchronous Task Processing.** ([Source Handover v4.0: 183])
    - For long-running operations within the /process endpoint (especially if batch sizes grow or document generation becomes more complex), consider

refactoring to use asynchronous task queues like Google Cloud Tasks to improve API responsiveness and scalability.

12. **Task: Admin User Approval UI.** ([Source Handover v4.0: 147])
    - 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.
13. **Task: Data Integrity for QuickBooks Lists (TERMS and SHIPVIA).** ([Source Handover v4.0: 164, 165])
    - If desired for more complete IIF files, implement functionality to ensure TERMS and SHIPVIA data matches QuickBooks lists exactly. This may involve UI changes for data entry or more complex backend mapping.
14. **Task: Automated Unit/Integration Tests.** ([Source Handover v4.0: 180])
    - Incrementally add automated tests (e.g., Pytest for backend API endpoints and service logic; Jest/React Testing Library for critical frontend components and logic) to improve code quality and ensure regressions are caught early.

8. Developer Setup & Environment (Recap from Handover Report v4.0, Section 11 & 12)

- **Backend:** Python 3.9+, virtual environment, pip install -r requirements.txt. Local .env file for secrets. GOOGLE_APPLICATION_CREDENTIALS pointing to Firebase Admin SDK service account key JSON for local development. Cloud SQL Auth Proxy for local DB connection.
- **Frontend:** Node.js (LTS), npm install. src/firebase.js (or similar) for Firebase project config. .env.development and .env.production for VITE_API_BASE_URL.
- **Deployment:**
    - Backend: Dockerized, pushed to Artifact Registry, deployed to Google Cloud Run.
    - Frontend: Built with npm run build, deployed to Firebase Hosting.
- Ensure all necessary IAM permissions are set for Cloud Run service accounts (e.g., to access Cloud SQL, GCS, Secret Manager, and invoke Firebase Admin SDK functions).

9. Conclusion for Handover

The G1 PO App is in a strong position, with the most complex piece of new functionality (multi-supplier PO processing with nuanced document generation and BigCommerce updates) now largely designed and implemented. The immediate next steps involve rigorously testing this new end-to-end flow, particularly the BigCommerce status updates and the accuracy of the new packing slip format.

Following that, completing the remaining MVP tasks like the Excel PO export will bring the application to a very capable state. The outlined future enhancements provide a clear roadmap for further development.

This report, combined with the previous detailed handover document (Version 4.0) and the codebase itself, should provide the incoming developer with all necessary information to successfully take over the project, complete the current objectives, and guide its future evolution.