G1 PO App - Definitive Handover Report & Project Plan

Date: May 17, 2025 (Reflecting all progress and chat sessions ending May 17, 2025, CDT)
Version: 11.0 (This document supersedes v10.0 and incorporates all recent development, bug fixes, and feature additions, including the "Standalone UPS Label Generator" and handling for "Customer-Billed UPS Shipments".)
Prepared For: Incoming Developer
Prepared By: Mark (via collaboration with AI Assistant Gemini)

Table of Contents

- AuthContext.jsx (Frontend)
- App.jsx (Frontend Routing)

5. **Database Schema Notes & Key Tables**
    - orders Table (Recent column additions)
    - shipments Table (Nullability change)
    - Other relevant tables (suppliers, purchase_orders, po_line_items, order_line_items, mapping tables)
6. **Developer Setup & Environment**
    - Backend
    - Frontend
    - Running Locally
    - Deployment
7. **Detailed Plan for Completion & Future Tasks (Updated & Prioritized)**
    - **Phase 1: Critical Post-Handover Tasks & Verification**
    - **Phase 2: Finalizing "Customer-Billed UPS Shipments" Feature**
    - **Phase 3: Standalone UPS Label Generator UI & Testing**
    - **Phase 4: MVP Task Completion (from original report, re-prioritized)**
    - **Phase 5: UI/UX Enhancements & Remaining Fixes (from original report, re-prioritized)**
    - **Phase 6: Infrastructure & Deployment Finalization (from original report, re-prioritized)**
    - **Phase 7: Long-Term Improvements & Best Practices (from original report, re-prioritized)**
8. **Known Issues & Considerations (Updated)**
9. **Recent Bug Fixes & Specific File Updates (Summarized from v10.0 and this chat)**
10. **Conclusion for Handover**

---

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.
Recent enhancements include:

- A "G1 Onsite Fulfillment" option for items fulfilled directly from stock.
- Functionality to handle orders where the customer uses their own UPS account for shipping charges.
- A standalone utility to generate UPS labels independent of the order processing workflow.

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 17, 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, UI enhancements, and the introduction of new fulfillment options.

Key Achievements & Functionalities (Consolidated from previous reports and this chat):

- **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.
    - **G1 Onsite Fulfillment:** UI and backend logic to process orders fulfilled directly from G1 stock (skips PO, generates Packing Slip and optional UPS Label, updates statuses, emails sales@globalonetechnology.com).
    - **Customer-Billed UPS Shipments:** Backend logic in ingest_orders parses customer notes for UPS account details and flags the order. shipping_service.py updated to attempt "Bill Third Party" to UPS.
- **Dashboard UI (Dashboard.jsx):**
    - Unified component handling "Orders" view and "Daily Sales Report" view.
    - "Daily Sales Report" integrated, accessible via navigation, fetches data from /api/reports/daily-revenue (UTC day aggregation), displays revenue for the last 14 UTC days with specific styling and timezone handling (hides current UTC day if sales are $0.00).
    - "Products" tab/UI removed from Dashboard.jsx; backend API routes for product mappings still exist.
- **Order Detail UI (OrderDetail.jsx - Original & Recent Enhancements):**
    - Prominent "ORDER PROCESSED SUCCESSFULLY!" message upon successful order processing.
    - Formats PO information as "PO # [Number] sent to [Supplier Name]".
    - Part number links (original_sku, spare_sku, hpe_option_pn) first copy Order Number to clipboard, then open Brokerbin. RFQ Sent status logic preserved.
    - Conditional "Use Multiple Suppliers" option based on line item count.
    - "G1 Onsite Fulfillment" option in supplier dropdown, with UI simplification when selected.
    - **Customer-Billed UPS Info Display:** Shows customer's UPS account if is_bill_to_customer_account is true. Default shipping method in processing forms now prioritizes customer_selected_freight_service for these orders.

- **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 for supplier POs.
    - Database interaction with PostgreSQL on Google Cloud SQL.
    - /api/reports/daily-revenue endpoint provides data for the Daily Sales Report.
    - /api/orders/<id>/process route significantly updated for G1 Onsite Fulfillment and now passes through data for customer-billed UPS shipments.
    - **New Standalone UPS Label**
      Route: /api/utils/generate_standalone_ups_label for generating UPS labels ad-hoc and emailing them to sales.
- **Document Generation (document_generator.py - ReportLab):**
    - Purchase Orders (PDFs): Includes supplier info, items, costs, notes, and a "Partial fulfillment" flag. COMPANY_ADDRESS_PO_HEADER is an empty string.
    - Packing Slips (PDFs): Differentiates items in current shipment vs. items shipping separately. Customer's phone number no longer included. Item descriptions use "pure" value. is_g1_onsite_fulfillment flag added to generate_packing_slip_pdf.
- **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.
    - **Customer-Billed UPS Shipments:** Updated generate_ups_label_raw to use "Bill Third Party" (Type "01") with the customer's UPS account number and their shipping postal code for validation when is_bill_to_customer_account is true.
- **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 (for yesterday's POs).
    - send_quickbooks_data_email function confirmed as unused and removed.
    - **New send_sales_notification_email function:** Implemented/verified for sending G1 Onsite Fulfillment notifications and Standalone UPS Labels to sales@globalonetechnology.com.
- **Cloud Storage (app.py - Google Cloud Storage):**
    - Uploads generated POs, Packing Slips, and Labels to GCS. Signed URLs are used for accessing these documents.
- **BigCommerce Integration (shipping_service.py, app.py):**
    - Retrieves order data.
    - Refined logic for shipment creation (now separate create_bigcommerce_shipment function in shipping_service.py) and final order status updates (separate set_bigcommerce_order_status function).
- **QuickBooks Integration (iif_generator.py, app.py):**
    - Generates and emails a daily IIF file for Purchase Orders (for "yesterday's" POs via scheduler, for "today's" POs via user trigger).
    - Corrected route in app.py for scheduled "yesterday's" batch: /api/tasks/scheduler/trigger-daily-iif-batch.
- **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.
    - Correct Firebase configuration and API key usage in frontend verified.
    - AuthContext.jsx updated to include an apiService for making authenticated backend calls.
- **GCS Signed URL IAM Permission:** "Service Account Token Creator" role for the Cloud Run service account for signed URL generation has been set up. *(Note: Still troubleshooting an issue where errors about needing a private key occur)*.

- **Routing (App.jsx):**
    - Main navigation bar links "Daily Sales" to /dashboard/sales.
    - /dashboard (and /) renders Dashboard component with initialView="orders".
    - /dashboard/sales route renders Dashboard component with initialView="dailySales".
    - Product-related routes removed.
    - **New Route:** /utils/standalone-label-generator added for the new utility.
- **Error Handling and Logging:** Implemented at various levels. Enhanced print(..., flush=True) and traceback.print_exc(file=sys.stderr) in backend routes for better Cloud Run logging.

Recent Developments & Enhancements (From This Chat Session - Post v10.0 Report):

- **Customer-Billed UPS Shipments Feature:**
    - Database orders table updated with customer_selected_freight_service, customer_ups_account_number, is_bill_to_customer_account, customer_ups_account_zipcode.
    - app.py (ingest_orders): Logic added to parse BigCommerce order comments for "|| Carrier: UPS || Service: ... || Account#: ..." to populate the new database fields.
    - app.py (process_order): Logic added to prioritize customer_selected_freight_service for label generation if is_bill_to_customer_account is true.
    - shipping_service.py (generate_ups_label_raw): Modified PaymentInformation to use "Bill Third Party" (Type "01") with the customer's UPS account number and their shipping postal code for validation if is_bill_to_customer_account is true.
    - OrderDetail.jsx:
        - Displays "Bill Shipping To: Customer UPS Account # ..." in Order Information if applicable.
        - Shipping method dropdowns in processing forms now default to customer_selected_freight_service if available for customer-billed orders.

- **Standalone UPS Label Generator Feature:**
    - Backend app.py: New route /api/utils/generate_standalone_ups_label created. It accepts ship-to details, package weight, and shipping method, calls shipping_service.generate_ups_label, and then emails the label to sales@globalonetechnology.com via email_service.send_sales_notification_email.
    - Frontend StandaloneUpsLabel.jsx: New component created with a form to input all necessary details.
    - Frontend App.jsx: Route and navigation link added for this new utility.
- **IIF Generator Routes Clarified:**
    - /api/tasks/scheduler/trigger-daily-iif-batch: Confirmed this route in app.py calls iif_generator.create_and_email_daily_iif_batch (for yesterday's POs) and is the target for the daily Cloud Scheduler job.
    - /api/tasks/trigger-iif-for-today-user: Confirmed this route calls iif_generator.create_and_email_iif_for_today (for today's POs) and is for user-triggering.
- **Email Service (email_service.py):**
    - Corrected attachment handling in send_po_email to expect already base64 encoded content and use the correct dictionary key 'Name'.
    - Confirmed send_sales_notification_email structure is suitable.
- **Bug Fixes Implemented During This Chat (also listed in v10.0 where applicable):**
    - shipping_service.py: Corrected state mapping in generate_ups_label_raw to handle 2-letter state codes.
    - Addressed various frontend/backend logging and error display nuances.

---

3. Technology Stack & Project Layout

- **Backend (order-processing-app directory):**
    - Python 3.9+
    - Flask (Web Framework)

    - SQLAlchemy (ORM for database interaction)
    - psycopg2-binary or pg8000 (PostgreSQL adapter, currently pg8000 via Cloud SQL Connector)
    - Gunicorn (WSGI Server for deployment)
    - ReportLab (PDF Generation)
    - Pillow (Image processing for label conversion)
    - python-dotenv (Environment variable management)
    - requests (HTTP requests)
    - google-cloud-sql-connector
    - google-cloud-storage
    - firebase-admin
    - postmarker (Postmark API client)
    - 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 build tool)
    - react-router-dom (Routing)
    - Firebase Client SDK (Authentication)
    - Standard CSS (and App.css, component-specific CSS).
    - Key components: App.jsx, Dashboard.jsx, OrderDetail.jsx, Login.jsx, AuthContext.jsx, ProtectedRoute.jsx, SupplierList.jsx, StandaloneUpsLabel.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 deployment).
    - Cloud SQL (PostgreSQL database).
    - Artifact Registry (Docker images).
    - Secret Manager (Secrets management).
    - Cloud Storage (GCS for documents).
    - Cloud Scheduler (IIF task automation).
    - Cloud Logging (Application and request logs).
- **Firebase:**
    - Firebase Hosting (Frontend deployment).
    - Firebase Authentication (User auth with Google Sign-In, Custom Claims).
- **External APIs:**
    - BigCommerce API (Orders, Products, Shipping).
    - UPS API (Shipping Labels - OAuth 2.0).
    - Postmark API (Transactional Emails).

---

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

- **app.py (Backend Flask Application):**
    - Provides all API endpoints.
    - **/api/ingest_orders:** Major updates to parse customer_message for || Carrier: UPS || Service: ... || Account#: ... and populate customer_selected_freight_service, customer_ups_account_number, and is_bill_to_customer_account in the orders table.
    - **/api/orders/<order_id>/process:**
        - Handles "G1 Onsite Fulfillment" and single/multi-supplier POs.

        - Updated to determine method_for_label_generation by prioritizing order_data_dict.customer_selected_freight_service if order_data_dict.is_bill_to_customer_account is true, before falling back to the method selected on the processing form.
        - Passes the full order_data_dict (which includes customer UPS account info) to shipping_service.generate_ups_label.
    - **/api/tasks/scheduler/trigger-daily-iif-batch:** Route confirmed/corrected to call iif_generator.create_and_email_daily_iif_batch (for YESTERDAY's POs).
    - **/api/tasks/trigger-iif-for-today-user:** Calls iif_generator.create_and_email_iif_for_today (for TODAY's POs).
    - **/api/utils/generate_standalone_ups_label (New):** Accepts shipping details, generates a UPS label via shipping_service, and emails it to sales via email_service.
    - Handles GCS uploads and Signed URL generation (still troubleshooting a "private key needed" error despite IAM permissions).
- **shipping_service.py (Backend):**
    - Manages UPS API interactions (OAuth, label generation) and BigCommerce shipment/status updates.
    - **generate_ups_label_raw:**
        - map_shipping_method_to_ups_code updated with more direct mappings for cleaner service names.
        - State processing logic updated to correctly handle 2-letter state codes or map full names.
        - **PaymentInformation block now dynamically constructed:**
            - If order_data.get('is_bill_to_customer_account') and order_data.get('customer_ups_account_number') are present, it uses Type: "01" and BillThirdParty with the customer's account number, their shipping postal code, and country code for validation.
            - Otherwise, defaults to Type: "01" and BillShipper (your account).

    - create_bigcommerce_shipment and set_bigcommerce_order_status are separate helper functions.
- **email_service.py (Backend):**
    - Handles sending emails via Postmark.
    - send_po_email: Attachment loop corrected to expect already base64 encoded content from app.py and use the correct 'Name' key.
    - send_iif_batch_email: Handles custom_subject for "today's" IIF.
    - send_sales_notification_email: Used by G1 Onsite and Standalone Label features.
    - send_quickbooks_data_email fully removed.
- **document_generator.py (Backend):**
    - Generates PO and Packing Slip PDFs using ReportLab.
    - generate_packing_slip_pdf accepts is_g1_onsite_fulfillment flag.
    - Logo fetched from GCS URI.
- **iif_generator.py (Backend):**
    - create_and_email_daily_iif_batch: Generates IIF for POs from *yesterday*.
    - create_and_email_iif_for_today: Generates IIF for POs from *today*.
    - Both use generate_iif_content_for_date as the core logic.
- **OrderDetail.jsx (Frontend):**
    - Displays order details and handles PO/fulfillment processing forms.
    - **Customer UPS Account Display:** Shows "Bill Shipping To: Customer UPS Account # ..." if applicable, including the service they selected from comments.
    - **Default Shipping Method:**
        - On initial load (fetchOrderAndSuppliers), the shipmentMethod state now defaults with priority:
          1. order.customer_selected_freight_service (if customer-billed),
          2. order.customer_shipping_method, 3. "UPS Ground".

        - When changing fulfillment mode (handleMainSupplierTriggerChange), the default method for the new mode also follows this priority.
        - For multi-supplier, newly assigned suppliers also get this prioritized default shipping method.
    - Uses apiService from AuthContext for backend calls.
- **StandaloneUpsLabel.jsx (Frontend) - New Component:**
    - Provides a form for users to input Ship To, Package, and Shipping Method details.
    - Submits data to /api/utils/generate_standalone_ups_label.
    - Displays success (with tracking #) or error messages.
- **AuthContext.jsx (Frontend):**
    - Manages Firebase authentication state.
    - **Includes apiService:** A utility (wrapper around fetch) that automatically attaches the Firebase ID token to outgoing requests to the backend API. Handles basic response parsing and error throwing.
- **App.jsx (Frontend Routing):**
    - Manages all application routes.
    - Includes a new route for /utils/standalone-label-generator pointing to StandaloneUpsLabel.jsx.

---

5. Database Schema Notes & Key Tables

- **orders Table:**
    - **Recent Additions:**
        - customer_selected_freight_service VARCHAR(100) NULL: Stores the shipping service name chosen by the customer if they use their own freight account (parsed from order comments).
        - customer_ups_account_number VARCHAR(20) NULL: Stores the customer's UPS account number if provided and if the carrier is UPS.

        - is_bill_to_customer_account BOOLEAN NULL DEFAULT FALSE: Flag set to true if the order should be billed to the customer's UPS account.
        - customer_ups_account_zipcode VARCHAR(15) NULL: Placeholder for the customer's UPS account billing ZIP, if it needs to be collected in the future for UPS validation. Currently not populated from BigCommerce.
- **shipments Table:**
    - purchase_order_id column: **Must be nullable** to support G1 Onsite Fulfillment shipments which do not have an associated purchase order.
    - order_id column: **Must be populated for all shipments**, including G1 Onsite and supplier PO shipments, linking back to the main orders table.
- **Other Key**
  Tables: suppliers, purchase_orders, po_line_items, order_line_items, hpe_part_mappings, hpe_description_mappings, qb_product_mapping. Their schemas are largely stable but ensure they support the current application logic.

---

6. Developer Setup & Environment

(This section is largely the same as v10.0, with minor clarifications)

- **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, SHIP_FROM_... details, etc.).
    - GOOGLE_APPLICATION_CREDENTIALS environment variable pointing to a Firebase Admin SDK service account key JSON file for local development (this key must belong to project g1-po-app-77790 and have necessary permissions like Firebase Admin, and ideally roles to impersonate or act as the Cloud Run runtime SA if needed for local GCS signing tests, though local GCS signing is tricky without the real Cloud Run environment's ADC).

    - 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).
    - Firebase project configuration snippet (API key, auth domain, etc.) must be correctly set up in src/firebase.js or similar, typically sourced 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, etc.
- **Running Locally:**
    - Start backend: flask run --port=8080 (or python app.py). Ensure .env is loaded.
    - Start frontend: npm run dev.
- **Deployment:**
    - **Backend:** Dockerized, pushed to Google Artifact Registry, deployed to Google Cloud Run.
        - The Cloud Run service account needs:
            - "Cloud SQL Client" role.
            - "Secret Manager Secret Accessor" role.
            - "Storage Object Admin" (or more granular read/write to the specific GCS bucket).
            - **"Service Account Token Creator" role ON ITSELF** (for GCS V4 Signed URLs).
            - Permissions to publish to Pub/Sub if using Cloud Tasks via Pub/Sub.
            - Any other roles needed for services it interacts with.

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

---

7. Detailed Plan for Completion & Future Tasks (Updated & Prioritized)

Phase 1: Critical Post-Handover Tasks & Verification (Some may be completed from v10.0)

1. **Database Schema Finalization (Critical - from v10.0, verify):**
    - **Action:** Confirm shipments.purchase_order_id is nullable and shipments.order_id is correctly foreign-keyed and populated for all shipment types.
    - **Action:** Confirm new columns in orders table (customer_selected_freight_service, customer_ups_account_number, is_bill_to_customer_account, customer_ups_account_zipcode) exist.
    - **File(s) Involved:** Database schema (SQL modification).
2. **Email Service send_sales_notification_email (Critical - from v10.0, verify):**
    - **Action:** Confirm email_service.py has send_sales_notification_email (or equivalent generic function) working correctly with attachments for G1 Onsite & Standalone Label features.
    - **File(s) Involved:** email_service.py, app.py (call sites).
3. **GCS Signed URL Generation (Ongoing Troubleshooting):**
    - **Action:** Resolve the "you need a private key to sign credentials" error.
        - Triple-confirm the Cloud Run service's *runtime service account*.
        - Ensure *that specific service account* has the "Service Account Token Creator" role granted *to itself*.
        - Redeploy the Cloud Run service after any IAM changes and wait for propagation.
        - If issues persist, examine app.py signed URL generation calls closely and consider advanced debugging like testing signBlob directly.
    - **File(s) Involved:** app.py, GCP IAM.
4. **Comprehensive End-to-End Testing of All Fulfillment Flows:**

- **Action:** After above items are stable, thoroughly test:
    - Single Supplier PO Workflow (including label billed to your account).
    - Multi-Supplier PO Workflow.
    - G1 Onsite Fulfillment Workflow.
    - **Customer-Billed UPS Shipment Workflow:** Test with a valid customer UPS account (ideally in UPS sandbox) that is authorized for recipient billing. Verify correct PaymentInformation is sent to UPS and that labels are generated.
    - Test edge cases (e.g., order with no items for G1 Onsite Fulfillment).
    - Verify all document content (POs, Packing Slips, Labels) and signed URL generation/access (once GCS issue is fixed).
- **File(s) Involved:** Full application stack.

Phase 2: Finalizing "Customer-Billed UPS Shipments" Feature

1. **UPS API Error 9120002 Resolution (Ongoing):**
    - **Action:** Continue testing the BillThirdParty (with Type: "01") structure in shipping_service.py, ensuring PostalCode (from recipient's shipping address) and CountryCode are included in BillThirdParty.Address.
    - **Action:** Work with UPS (sandbox/developer support) to ensure the test account numbers used are valid for this billing type and to understand any specific validation requirements (e.g., if payer's *actual billing ZIP* is strictly required instead of recipient's shipping ZIP).
    - **File(s) Involved:** shipping_service.py.
2. **Collect Customer's UPS Account Billing ZIP (If Necessary):**
    - **Action:** If UPS mandates the payer's *actual* billing ZIP for validation (and recipient's shipping ZIP is not sufficient), plan to:
        - Modify the BigCommerce checkout script to add a field for "Billing ZIP for your UPS Account".
        - Update app.py (ingest_orders) to parse and store this in orders.customer_ups_account_zipcode.

        - Update shipping_service.py to use this specific ZIP in the BillThirdParty.Address.PostalCode.
    - **File(s) Involved:** BigCommerce checkout script, app.py, shipping_service.py.
3. **Enhanced Error Feedback for Customer-Billed Failures:**
    - **Action:** Modify shipping_service.py to return more specific error details if UPS rejects customer account billing.
    - **Action:** Modify app.py (process_order) to catch these specific errors and return a more informative 4xx error to the frontend.
    - **Action:** Update OrderDetail.jsx to display these specific errors to the user.
    - **File(s) Involved:** shipping_service.py, app.py, OrderDetail.jsx.

Phase 3: Standalone UPS Label Generator UI & Testing

1. **Frontend UI Implementation (StandaloneUpsLabel.jsx):**
    - **Action:** Review and finalize the StandaloneUpsLabel.jsx component and its CSS.
    - Ensure all form fields, validations, and API calls to /api/utils/generate_standalone_ups_label are robust.
    - **File(s) Involved:** StandaloneUpsLabel.jsx, StandaloneUpsLabel.css, App.jsx (for routing).
2. **End-to-End Testing:**
    - **Action:** Thoroughly test the standalone label generation for various valid and invalid inputs.
    - Confirm email delivery with correct label attachment to sales@globalonetechnology.com.
    - Confirm correct use of SHIP_FROM_... details from backend environment variables.
    - **File(s) Involved:** Full stack for this feature.

Phase 4: MVP Task Completion (from original report, re-prioritize as needed)

(These are from the original v9.0/v10.0 plan, re-evaluate priority)

6. **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.

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

* Frontend error handling for 401/403.
* Documentation for setAdminClaim.cjs (or prioritize Admin UI for claim management - see Phase 7).
* Re-verify Cloud Run IAM permissions comprehensively.

Phase 5: UI/UX Enhancements & Remaining Fixes (from original report, re-prioritized)

1. **Task: Supplier and Product/Mapping Management UI.**
    - Implement React components for CRUD operations on suppliers.
    - Re-evaluate need/priority for UI for hpe_part_mappings, hpe_description_mappings, qb_product_mapping, products. If needed, design an entry point (e.g., Admin section).
2. **Task: Frontend - Eloquia Font Issue.**
    - Resolve font rendering on the deployed site. (Check @font-face in App.css, font file paths in public folder, and build process).
3. **Task: Loading Indicators and UX Refinements.**
    - Audit API-calling components for consistent loading indicators.
    - Review UX of the new G1 Onsite Fulfillment flow and Customer-Billed Shipment indications in OrderDetail.jsx.
    - Review UX of the Standalone UPS Label generator.

Phase 6: Infrastructure & Deployment Finalization (from original report, re-prioritized)

1. **Task: Postmark - External Domain Sending Approval.**
    - Ensure Postmark is fully configured (DKIM, SPF, Custom Return-Path) for reliable email delivery from your domain.
2. **Task: Cloudflare DNS for Production Domains (If applicable).**
    - Configure g1po.com (frontend) and api.g1po.com (backend) if using custom domains.

Phase 7: Long-Term Improvements & Best Practices (from original report, re-prioritized)

1. **Task: Backend Daily Revenue Aggregation Timezone.**
    - 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.
2. **Task: Security Hardening.**
    - Cloud Run Ingress (consider IAP - Identity-Aware Proxy).
    - Comprehensive server-side input validation for all API endpoints.
    - Review and enhance structured, detailed, leveled logging.
3. **Task: Asynchronous Task Processing.**
    - For /api/orders/<id>/process, especially document generation and external API calls, consider using Google Cloud Tasks to improve API response times and robustness.
4. **Task: Admin User Approval UI.**
    - Implement an admin interface for managing isApproved Firebase custom claims, replacing the manual setAdminClaim.cjs script.
5. **Task: Data Integrity for QuickBooks Lists (TERMS and SHIPVIA).**
    - Ensure data used for IIF generation matches QuickBooks lists precisely.
6. **Task: Automated Unit/Integration Tests.**
    - Implement a testing suite using Pytest (backend) and Jest/React Testing Library (frontend).

---

8. Known Issues & Considerations (Updated)

- **GCS Signed URL Generation:** Persisting issue with "you need a private key" error despite "Service Account Token Creator" role being set on the Cloud Run service account. Requires focused troubleshooting on service account identity and IAM propagation.

- **UPS Customer Account Billing (Error 9120002):** Still unresolved. Requires testing with BillThirdParty structure including PostalCode and CountryCode, and verifying test account validity/authorization with UPS for this billing type. May require collecting payer's specific billing ZIP if recipient's shipping ZIP is insufficient.
- **IIF Import Sensitivities:** QuickBooks Desktop IIF import is sensitive. Thorough testing of generated IIF files is crucial.
- **UPS Production Costs:** Live UPS API will incur costs. Ensure proper testing in the UPS test environment before going live.
- **Postmark Deliverability:** DNS setup (DKIM, SPF, Return-Path) is critical for email deliverability.
- **Scalability of Synchronous Processing:** Current order processing in /api/orders/<id>/process is synchronous. For higher volumes, consider moving to asynchronous tasks (see Phase 7).
- **Environment Management:** Maintain strict separation and configuration for development, staging (if any), and production environments.
- **Custom Claim Management:** Admin UI for Firebase custom claims (isApproved) is a future task; currently manual via setAdminClaim.cjs.
- **Firebase & GCP Quotas:** Monitor usage of Firebase services and GCP resources.
- **Service Account Key Security (Local Dev):** Protect any downloaded service account key JSON files diligently. Use them only for local development and prefer workload identity federation or service account impersonation in deployed environments where possible.
- **Timezone for Reporting:** The Daily Sales Report currently uses UTC day aggregation. Backend changes may be needed for local business day alignment if required (Phase 7).
- **BigCommerce API Rate Limits:** Be mindful of BigCommerce API rate limits, especially during bulk operations like order ingestion. Implement retries with backoff if necessary.
- **Order Comment Parsing Reliability:** Parsing freight details from customer_message is functional but relies on a specific string format. Consider making this more robust or using BigCommerce Order Metafields if the current script becomes hard to maintain or error-prone due to comment length or format variations.

9. Recent Bug Fixes & Specific File Updates (Summarized from v10.0 and this chat)

- **app.py:**
    - ingest_orders: Added parsing for || Carrier: UPS ... in customer_message to populate new orders table columns for customer-billed shipping.

      Conditional 敏感词 stripping.
    - process_order: Updated to use method_for_label_generation logic, passes full order_data_dict to label generation. Added order_id to supplier PO shipment inserts.
    - New route /api/utils/generate_standalone_ups_label.
    - Corrected route /api/tasks/scheduler/trigger-daily-iif-batch.
    - Added detailed print(..., flush=True) and traceback.print_exc(file=sys.stderr) for debugging.
- **shipping_service.py:**
    - generate_ups_label_raw:
        - Corrected state mapping to handle 2-letter codes.
        - Updated PaymentInformation to use BillThirdParty with Type: "01", AccountNumber, Address.PostalCode (from recipient shipping ZIP), and Address.CountryCode when is_bill_to_customer_account is true.
    - map_shipping_method_to_ups_code: Added more direct string matches for cleaner service names.
- **email_service.py:**
    - send_po_email: Corrected attachment loop to expect pre-base64 encoded content and use Name key.
    - send_quickbooks_data_email completely removed.
- **OrderDetail.jsx:**
    - Displays customer UPS account info and selected service if applicable.

    - Processing form shipping method dropdowns now default based on customer_selected_freight_service (if customer-billed) > order.customer_shipping_method > "UPS Ground".
    - Uses apiService from AuthContext.
- **StandaloneUpsLabel.jsx:** New component created.
- **AuthContext.jsx:** Updated to include apiService for authenticated backend calls.
- **Database:**
    - orders table: Added customer_selected_freight_service, customer_ups_account_number, is_bill_to_customer_account, customer_ups_account_zipcode.
    - shipments table: purchase_order_id confirmed nullable, order_id to be populated for all.

---

10. Conclusion for Handover

The G1 PO App has significantly evolved, incorporating advanced fulfillment options like G1 Onsite, customer-billed UPS shipments, and a standalone label utility. The backend has been updated to support these features, and the frontend reflects many of these changes.

Immediate priorities for the incoming developer are to resolve the GCS Signed URL generation issue and finalize the UPS "Bill Recipient/Third Party" functionality by addressing the API error 9120002. Following this, a comprehensive testing phase is crucial. The detailed plan above outlines further steps for completing MVP features, UI/UX refinements, and long-term improvements.