G1 PO App - Consolidated Handover Report & Project Plan (v14.0)

Date: May 20, 2025 (Reflecting all progress and chat sessions ending May 20, 2025, CDT)
Version: 14.0 (This document supersedes v13.0 and incorporates all subsequent development, troubleshooting, decisions, and planning from our recent chat session.)

Prepared For: Incoming Developer
Prepared By: AI Assistant Gemini (in collaboration with Mark)

Objective: This document provides a comprehensive overview of the "G1 PO App," its current state, architecture, recent progress, known issues, and a detailed plan for future development and completion. It is intended to facilitate a smooth handover and continued development.

Table of Contents

5. **Database Schema Notes & Key Tables (Including recent additions)**
6. **Developer Setup & Environment**
    - Backend
    - Frontend
    - Running Locally
    - Deployment (Cloud Run & Firebase Hosting)
    - Crucial Environment Variables
7. **Detailed Plan for Completion & Future Tasks (Consolidated, re-prioritized, and incorporating recent developments)**
    - Phase 0: Critical Immediate Issues (Post-Handover) - *Mostly Resolved or Mitigated*
    - Phase 1: FedEx Integration Finalization - *Partially Blocked*
    - **Phase 2: QuickBooks Integration Module (IIF-Based - Current Focus)**
    - Phase 3: UI/UX Enhancements & Remaining Fixes
    - Phase 4: MVP Task Completion
    - Phase 5: Infrastructure & Deployment Finalization
    - Phase 6: Long-Term Improvements & Best Practices (including potential QBWC exploration)
8. **Known Issues & Considerations (Updated & Consolidated)**
9. **Summary of Recent Changes (From This Chat Session - May 20, 2025)**
10. **Conclusion for Handover & Immediate Next Steps for Human Developer**

---

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 and G1-stocked items. The primary aim is to reduce manual effort, improve accuracy, provide a centralized platform for critical business workflows, and enhance operational efficiency,

including replacing functionalities of tools like T-HUB for BigCommerce-to-QuickBooks synchronization.

Core Functionality:

- Ingest orders from BigCommerce.
- Allow authorized users to manage these orders via a web interface.
- Generate necessary documentation (POs, Packing Slips, Shipping Labels for UPS & FedEx).
- Facilitate communication with suppliers (e.g., emailing POs).
- Update BigCommerce with shipment details and order statuses.
- Integrate with QuickBooks Desktop for accounting (Purchase Orders and Sales Orders/Invoices & Payments via IIF files).

---

2. Current State of the Application (As of May 20, 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 and administrative utilities. The current focus has shifted heavily towards implementing and refining the QuickBooks Desktop integration using IIF files.

- **Key Achievements & Functionalities (Consolidated & Updated from v13.0)**
    - **Order Ingestion & Management:**
        - Successfully ingests relevant orders from BigCommerce (filtered by status, e.g., "Awaiting Fulfillment")
          via app.py's ingest_orders function.
        - Parses BigCommerce customer notes for UPS and FedEx third-party billing details (Carrier, Service, Account #).
        - Stores ingested BigCommerce billing_address components and shipping_cost_ex_tax into dedicated columns in the orders table.
        - Preserves manually set statuses in the local DB and strips sensitive patterns from customer notes.

        - Dashboard for viewing/filtering orders (by status, with status counts) 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 vs. G1 Onsite Fulfillment.
    - **Fulfillment Modes:**
        - **Single/Multi-Supplier POs:** Creates distinct Purchase Orders, PO Line Items, and Shipment records. Generates unique, sequential PO numbers.
        - **G1 Onsite Fulfillment:** UI and backend logic to process orders from G1 stock (skips PO, generates Packing Slip and optional UPS Label, updates statuses, emails sales@globalonetechnology.com).
        - **Customer-Billed Shipments (UPS & FedEx):** Backend logic in ingest_orders parses customer notes for carrier account details and flags the order. shipping_service.py updated to attempt "Bill Third Party" (UPS) or "Bill RECIPIENT" (FedEx). OrderDetail.jsx displays customer's carrier account info.
    - **Dashboard UI (Dashboard.jsx):** Unified component handling "Orders" view and "Daily Sales Report" view. "Daily Sales Report" integrated, fetches data from /api/reports/daily-revenue (UTC day aggregation), displays revenue for the last 14 UTC days.
    - **Order Detail UI (OrderDetail.jsx):** Prominent "ORDER PROCESSED SUCCESSFULLY!" message. Formats PO info. Part number links copy Order Number to clipboard, then open Brokerbin. "RFQ Sent" status logic integrated. Input lag issue in PO item entry fields resolved.
    - **Backend Processing (app.py - Python/Flask):** Handles all order processing requests, multi-supplier logic, G1 Onsite fulfillment. Passes appropriate data for customer-billed UPS/FedEx shipments.
      Includes /api/utils/generate_standalone_ups_label route.
    - **Document Generation (document_generator.py - ReportLab, Pillow):**

        - **Purchase Orders (PDFs):** Includes supplier info, items, costs, notes, "Partial fulfillment" flag.
        - **Packing Slips (PDFs):** Differentiates items in current shipment vs. items shipping separately. is_g1_onsite_fulfillment flag for tailored content. Customer phone number removed. Payment method display formats "ElementA (ElementB)" to "ElementB".
        - Logo fetching from GCS.
    - **Shipping Label Generation (shipping_service.py):**
        - **UPS (OAuth 2.0):**
            - Fully functional for "Bill Sender" (using older PaymentInformation for account EW1847) and "Bill Third Party" (customer account, using current PaymentDetails).
            - Generates PDF labels (via GIF conversion), 1-inch margins, top-aligned.
            - *Outstanding Issue (RESOLVED in v13.0 conclusion, re-verify):* Intermittent missing label image bytes from UPS. Enhanced logging added.
        - **FedEx (OAuth 2.0):**
            - OAuth 2.0 token retrieval for sandbox AND production functional (CXS-TP scope).
            - Service code mapping implemented.
            - generate_fedex_label_raw() and generate_fedex_label() drafted .
            - "Bill SENDER" successfully tested against FedEx PRODUCTION (test-marked label).
            - *FedEx "Bill RECIPIENT" sandbox testing BLOCKED (ACCOUNT.VALIDATION.FAILED).* Awaiting FedEx guidance/test accounts.
            - *FedEx Production Label Certification is PENDING.*
    - **Email Communication (email_service.py - Postmark API):** Emails PO PDF, Packing Slip PDF, and Shipping Label PDF (if generated) to

suppliers. send_sales_notification_email for G1 Onsite/Standalone UPS labels. Sends daily IIF batch emails.

- **Cloud Storage (app.py - Google Cloud Storage):** Uploads generated POs, Packing Slips, and Labels to GCS.
    - *Signed URL generation issue (RESOLVED in v13.0 conclusion, re-verify)*.
- **BigCommerce Integration (shipping_service.py, app.py):** Retrieves order data. Refined logic for shipment creation and final order status updates.
- **QuickBooks Desktop Integration (IIF - iif_generator.py, app.py):**
    - Generates and emails a daily IIF file for **Purchase Orders** (yesterday's via scheduler, today's via user trigger).
    - **NEW (This Chat):** Logic added to iif_generator.py to generate IIF files for **Sales Orders (as Invoices) and corresponding Payments** for all pending orders.
        - Uses "G1.com Website Customer" for all sales.
        - Invoice DOCNUM is bigcommerce_order_id.
          Payment DOCNUM is also bigcommerce_order_id.
        - Dates are converted to US Central Time for IIF output.
        - Item descriptions for sales invoices
          use hpe_description_mappings.po_description with fallback to product name.
        - Shipping line item description uses actual shipping method, or "UPS Ground" for "free shipping", and handles "ElementA (ElementB)" format.
        - Billing address from orders table is used for
          Invoice ADDR fields.
        - Handles US state name to 2-letter abbreviation and adds comma after city in addresses.
    - **NEW (This Chat):** iif_generator.py functions now return a list of processed DB IDs to app.py.

        - **NEW (This Chat):** /api/quickbooks/trigger-sync endpoint in app.py created to:
            - Call iif_generator.py for all pending POs and Sales Orders.
            - Email the generated IIF files.
            - Update qb_po_sync_status and qb_sales_order_sync_status t o 'synced' in the database for successfully processed records (using the returned DB IDs).
    - **Authentication & Authorization (Firebase):** Robust user authentication (Google Sign-In). Authorization using Firebase Custom Claims (isApproved: true). Backend API routes protected. AuthContext.jsx provides apiService.
    - **Frontend UI Structure & Utilities (App.jsx, UtilitiesLandingPage.jsx, etc.):** Navigation: "Orders | Daily Sales | Utilities". Utilities page links to Standalone UPS Label Generator (functional), QuickBooks Sync (QuickbooksSync.jsx - recently built), Supplier Management, HPE Description Management. Dark Mode CSS implemented. CORS issues resolved. Supplier Management UI routing and apiService integration fixed.
- **Recent Developments & Enhancements (From This Chat Session - May 20, 2025)**
    - **FedEx Label Testing:** Confirmed shipping_service.py is ready for "Bill SENDER" FedEx Priority Overnight. Production test resulted in 403 (likely due to pending label certification). Sandbox test for Priority Overnight failed with SERVICETYPE.NOTSUPPORTED (sandbox account limitation). Sandbox test for FedEx Ground **succeeded**.
    - **QuickBooks IIF - Sales Orders & Payments:**
        - iif_generator.py extensively updated to generate IIF for Sales (Invoices + Payments).
        - Logic added for US Central Time date formatting, state name to abbreviation, comma after city in addresses, specific item description sources, and shipping line description formatting.
        - Billing address from orders table is now used.
        - iif_generator.py functions modified to return lists of processed DB IDs.
    - **QuickBooks IIF - Sync Status & Backend API:**

        - app.py's ingest_orders function updated to correctly fetch and store BigCommerce billing_address components and shipping_cost_ex_tax into the orders table.
        - New endpoint /api/quickbooks/trigger-sync created in app.py. This endpoint:
            - Calls iif_generator.py to generate IIFs for ALL pending POs and Sales Orders.
            - Emails these IIFs.
            - Updates qb_po_sync_status and qb_sales_order_sync_status to 'synced' for records successfully included in the emailed batches, using the returned DB IDs.
    - **QuickBooks IIF - Frontend:**
        - Basic QuickbooksSync.jsx component created with a button to trigger the /api/quickbooks/trigger-sync endpoint.
        - CSS for QuickbooksSync.jsx updated to use variables from FormCommon.css for theme consistency.
        - Resolved frontend TypeError: response.json is not a function by correcting how QuickbooksSync.jsx handles the promise from apiService.
    - **IIF Data Issues Resolved:**
        - Corrected sign convention for AMOUNT on Invoice SPL lines (now negative).
        - Modified sanitize_field in iif_generator.py to replace " with "in" to prevent IIF import errors for PO descriptions.
    - **General Debugging:** Addressed various Python errors (indentation, undefined variables, incorrect function arguments) in app.py and iif_generator.py.

---

3. Technology Stack & Project Layout (Largely from v13.0, minor updates noted)

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

- **Language/Framework:** Python 3.9+, Flask
- **ORM/Database:** SQLAlchemy, psycopg2-binary (or pg8000 via Cloud SQL Connector) for PostgreSQL.
- **WSGI Server:** Gunicorn (for deployment)
- **Key Libraries:**
  - requests (HTTP calls to BigCommerce, UPS, FedEx, Postmark)
  - python-dotenv (Environment variable management)
  - ReportLab, Pillow (PDF/Image generation)
  - google-cloud-sql-connector, google-cloud-storage (GCP integration)
  - firebase-admin (Authentication, Custom Claims)
  - postmarker (Postmark API client)
  - pytz (for timezone conversions, **newly added for IIF dates**)
- **Dependencies:** requirements.txt
- **Configuration:** .env (local), Google Cloud Secret Manager (deployed)

- **Frontend (order-processing-app-frontend directory)**
  - **Framework/Library:** React (Vite build tool)
  - **Routing:** react-router-dom
  - **Authentication:** Firebase Client SDK
  - **Styling:** Standard CSS, component-specific CSS files (e.g., OrderDetail.css, FormCommon.css, QuickbooksSync.css).
  - **Key Components:** App.jsx, Dashboard.jsx, OrderDetail.jsx, Login.jsx, AuthContext.jsx, ProtectedRoute.jsx, UtilitiesLandingPage.jsx, SupplierList.jsx, SupplierForm.jsx, EditSupplierForm.jsx, HpeDescriptionList.jsx, HpeDescriptionForm.jsx, EditHpeDescriptionForm.jsx, StandaloneUpsLabel.jsx, QuickbooksSync.jsx (**newly built**).
  - **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 image storage.
    - **Secret Manager:** Secrets management.
    - **Cloud Storage (GCS):** Document storage (POs, Packing Slips, Labels).
    - **Cloud Scheduler:** IIF task automation (for daily "yesterday's" batches).
    - **Cloud Logging:** Application and request logs.
    - **IAM:** Identity and Access Management.
- **Authentication**
    - **Firebase Authentication:** User authentication (Google Sign-In), Firebase Custom Claims (isApproved: true).
- **External APIs**
    - **BigCommerce API:** Orders, Products, Shipping.
    - **UPS API:** Shipping Labels (OAuth 2.0).
    - **FedEx API:** Shipping Labels (OAuth 2.0).
    - **Postmark API:** Transactional Emails.

---

4. Key Code Files & Their Roles

- **Backend (order-processing-app directory):**
    - **app.py (Flask Application):**
        - Provides all API endpoints for orders, suppliers, HPE descriptions, lookups, utilities, scheduled tasks, and reports.
        - Handles order ingestion (/api/ingest_orders) including parsing customer notes and storing BC billing address and shipping cost.

        - Main order processing logic in /api/orders/<order_id>/process.
        - CRUD endpoints for Suppliers and HPE Description Mappings.
        - Standalone UPS label generation route.
        - **NEW:** /api/quickbooks/trigger-sync endpoint to initiate on-demand IIF generation for all pending POs and Sales Orders, email them, and update their DB sync statuses.
        - Protected by @verify_firebase_token decorator.
    - **shipping_service.py:** Manages UPS and FedEx API interactions (OAuth, label generation). Contains logic for "Bill Sender" and "Bill Third Party/Recipient". Includes GIF to PDF conversion with margin adjustments.
    - **document_generator.py:** Generates Purchase Order and Packing Slip PDFs using ReportLab. Handles logo fetching from GCS. Packing slip tailors content for G1 Onsite Fulfillment and formats payment method display.
    - **email_service.py:** Manages sending emails via Postmark API (POs, Packing Slips, Labels, IIF batches, Sales Notifications). send_iif_batch_email function updated to handle filename_prefix.
    - **iif_generator.py:**
        - generate_po_iif_content_for_date(): Generates IIF content for Purchase Orders. Filters by po.status = 'SENT_TO_SUPPLIER' and qb_po_sync_status. Converts dates to US Central Time. Returns IIF string, mapping failures, and list of processed PO DB IDs.
        - generate_sales_iif_content_for_date(): **NEW/EXTENSIVELY MODIFIED**. Generates IIF content for Sales Orders (Invoices) and corresponding Payments. Filters by qb_sales_order_sync_status. Converts dates to US Central Time.
          Uses hpe_description_mappings for item descriptions. Formats shipping line description. Uses billing address from orders table. Handles US state abbreviations and city comma. Returns IIF string, mapping failures, and list of processed Sales Order DB IDs.
        - Top-level functions (create_and_email_...) for daily and on-demand IIF generation (these call the core generation functions).

- **Frontend (order-processing-app-frontend directory):**
    - **AuthContext.jsx:** Manages Firebase auth state and provides apiService for authenticated backend calls. apiService correctly handles JSON responses and errors.
    - **QuickbooksSync.jsx: NEW**. UI component with a button to call /api/quickbooks/trigger-sync. Displays status messages from the backend. CSS aligned with FormCommon.css.
    - OrderDetail.jsx: Displays order details, allows processing. UI adapts to fulfillment mode.
    - FormCommon.css: Centralized CSS for forms, including light/dark mode variables.

---

5. Database Schema Notes & Key Tables (Including recent additions)

(Consolidated from v13.0, with updates from this session)

- **orders Table:**
    - Stores ingested BigCommerce order details.
    - Key columns: id, bigcommerce_order_id, customer_name, customer_company, shipping address fields (customer_shipping_...), customer_shipping_method, customer_notes, status (local app status: 'new', 'RFQ Sent', 'Processed', 'international_manual', 'pending', 'Completed Offline'), is_international, payment_method, total_sale_price, bigcommerce_order_tax, order_date.
    - **UPS Customer Billing**
      Fields: customer_selected_freight_service, customer_ups_account_number, is_bill_to_customer_account, customer_ups_account_zipcode.
    - **FedEx Customer Billing**
      Fields: customer_selected_fedex_service, customer_fedex_account_number, is_bill_to_customer_fedex_account, customer_fedex_account_zipcode.
    - **NEW (from this session):**

        - bc_shipping_cost_ex_tax (DECIMAL): Stores shipping_cost_ex_tax from BigCommerce.
        - customer_billing_first_name (VARCHAR)
        - customer_billing_last_name (VARCHAR)
        - customer_billing_company (VARCHAR)
        - customer_billing_street_1 (VARCHAR)
        - customer_billing_street_2 (VARCHAR)
        - customer_billing_city (VARCHAR)
        - customer_billing_state (VARCHAR)
        - customer_billing_zip (VARCHAR)
        - customer_billing_country (VARCHAR)
        - customer_billing_country_iso2 (VARCHAR(2))
        - customer_billing_phone (VARCHAR)
    - **QuickBooks Sync Status (Phase 2):**
        - qb_sales_order_sync_status (VARCHAR: 'pending_sync', 'synced', 'error', 'skipped', DEFAULT 'pending_sync')
        - qb_sales_order_synced_at (TIMESTAMP WITH TIME ZONE)
        - qb_sales_order_last_error (TEXT)
- **order_line_items Table:** Stores individual line items for each order. Links to orders table.
- **suppliers Table:** Stores supplier information.
- **purchase_orders Table:**
    - Stores generated Purchase Orders.
    - Key columns: id, po_number (sequentially generated), order_id, supplier_id, po_date, status ('New', 'SENT_TO_SUPPLIER'), total_amount, payment_instructions, po_pdf_gcs_path, packing_slip_gcs_path.
    - **QuickBooks Sync Status (Phase 2):**

        - qb_po_sync_status (VARCHAR, DEFAULT 'pending_sync')
        - qb_po_synced_at (TIMESTAMP WITH TIME ZONE)
        - qb_po_last_error (TEXT)
- **po_line_items Table:** Stores line items for each Purchase Order.
- **shipments Table:** Stores shipment details (tracking, label paths). purchase_order_id is NULLABLE (for G1 Onsite Fulfillment).
- **hpe_part_mappings Table:** Maps original SKUs (from BC order line items) to HPE Option PNs and Spare PNs. Used for PO item descriptions and Sales Invoice item lookups.
- **hpe_description_mappings Table:** Stores custom PO descriptions for HPE Option PNs. Actively managed via UI. Used for PO and Sales Invoice item descriptions.
- **qb_product_mapping Table:** Maps option_pn (from hpe_part_mappings or direct SKU) to qb_item_name (QuickBooks Item Name/Number). Used for PO and Sales Invoice IIF generation.
- **users Table (Implicit via Firebase):** User authentication (Firebase UID, email) and authorization (isApproved custom claim) are managed in Firebase.

---

6. Developer Setup & Environment (Consolidated from v13.0)

- **Backend (order-processing-app directory):**
    1. **Python:** Version 3.9+ recommended.
    2. **Virtual Environment:** Create and activate (e.g., python -m venv venv, source venv/bin/activate or venv\Scripts\activate).
    3. **Dependencies:** pip install -r requirements.txt (ensure pytz is added).
    4. **Environment Variables (.env file):** Create in order-processing-app directory.
        - Database: DB_CONNECTION_NAME, DB_USER, DB_PASSWORD, DB_NAME, DB_DRIVER (e.g., pg8000).
        - BigCommerce: BIGCOMMERCE_STORE_HASH, BIGCOMMERCE_ACCESS_TOKEN, BC_PROCESSING_STATUS_ID, BC_SHIPPED_STATUS_ID, DOMESTIC_COUNTRY_CODE.

- UPS
  API: UPS_CLIENT_ID, UPS_CLIENT_SECRET, UPS_BILLING_ACCOUNT_NUMBER (EW1847), UPS_API_ENVIRONMENT (production/test), UPS_API_VERSION.
- FedEx
  API: FEDEX_API_KEY_SANDBOX, FEDEX_SECRET_KEY_SANDBOX, FEDEX_ACCOUNT_NUMBER_SANDBOX, FEDEX_OAUTH_URL_SANDBOX, FEDEX_SHIP_API_URL_SANDBOX. Also corresponding _PRODUCTION variables. FEDEX_API_ENVIRONMENT (sandbox/production), FEDEX_GRANT_TYPE.
- Postmark: POSTMARK_SERVER_TOKEN (was EMAIL_API_KEY), EMAIL_SENDER_ADDRESS, EMAIL_BCC_ADDRESS, QUICKBOOKS_EMAIL_RECIPIENT, DAILY_IIF_EMAIL_SUBJECT_PREFIX.
- GCS: GCS_BUCKET_NAME, COMPANY_LOGO_GCS_URI.
- Ship From
  Address: SHIP_FROM_NAME, SHIP_FROM_CONTACT, SHIP_FROM_STREET1, etc.
- Firebase/GCP: GOOGLE_APPLICATION_CREDENTIALS (path to service account key JSON for local Firebase Admin SDK & GCS).
- Flask: FLASK_DEBUG (True/False).

5. **Google Cloud SQL Auth Proxy:** Required for local connection to Cloud SQL PostgreSQL. Download, configure, and run.
6. **Run Backend:** python app.py (or flask run --host=... --port=... as per your .env or defaults).

- **Frontend (order-processing-app-frontend directory):**
  1. **Node.js:** LTS version recommended.
  2. **Dependencies:** npm install (or yarn install).
  3. **Environment Variables (.env.development and/or .env.local):**
     - VITE_API_BASE_URL: Base URL of your local backend (e.g., http://127.0.0.1:8080/api). Ensure it ends with /api.

        - Firebase SDK
          Config: VITE_FIREBASE_API_KEY, VITE_FIREBASE_AUTH_DOMAIN, etc. (from Firebase project settings).
    4. **Run Frontend:** npm run dev.
- **Deployment:**
    - **Backend (Cloud Run):** Dockerized, pushed to Google Artifact Registry, deployed to Cloud Run.
        - **CRITICAL Cloud Run Service Account IAM Roles:** Cloud SQL Client, Secret Manager Secret Accessor, Storage Object Admin (or granular), Service Account Token Creator (granted TO ITSELF for GCS V4 Signed URLs).
    - **Frontend (Firebase Hosting):** npm run build, then firebase deploy --only hosting.

---

7. Detailed Plan for Completion & Future Tasks

(Consolidated, re-prioritized, incorporating recent developments. Tasks from v13.0 Phase 0 and 1 are mostly addressed or their status clarified).

- **Immediate Focus (Post-Handover Validation & Refinement):**
    1. **Thoroughly Test QuickBooks IIF for Sales Orders/Payments (Manual Import):**
        - **Action:** Generate IIF files for sales orders (using python iif_generator.py with process_all_pending=True or for specific test dates).
        - **Verify:** Import into a TEST QuickBooks Desktop company file. Check:
            - Invoice creation for "G1.com Website Customer".
            - Correct DOCNUM (BC Order ID), DATE (US Central Time).
            - Correct TERMS ("Prepay-Credit Card").
            - Correct Bill To (ADDR fields - using billing address from orders table) and Ship To (SADDR fields).

        - Line items:
          Correct INVITEM (from qb_product_mapping), QNTY, PRICE, A MOUNT (negative for
          income), MEMO (from hpe_description_mappings).
        - Shipping line: Correct INVITEM ("Freight
          Collected"), AMOUNT (negative), MEMO (shipping method name / "UPS Ground").
        - Tax line: Correct INVITEM ("Sales Tax"), ACCNT ("Other Income"), AMOUNT (negative).
        - Payment creation: Correct ACCNT ("Undeposited
          Funds"), NAME, AMOUNT, DOCNUM (BC Order ID), PAYMETH.
        - Payment SPL line: Correctly credits "Accounts Receivable", correct AMOUNT (negative).
        - Ensure payment is applied to the invoice in QuickBooks.
    - **Files:** iif_generator.py, Test QuickBooks Company File.
2. **Thoroughly Test QuickBooks IIF for Purchase Orders (Manual Import):**
    - **Action:** Generate IIF for POs.
    - **Verify:** Import into TEST QB. Check all fields, especially supplier name, item descriptions, costs, and notes.
    - **Files:** iif_generator.py.
3. **Refine Backend /api/quickbooks/trigger-sync Status Updates in app.py:**
    - **Action:** Ensure generate_po_iif_content_for_date and generate_sales _iif_content_for_date in iif_generator.py correctly return the lists of processed database IDs.
    - **Action:** Modify the trigger_quickbooks_sync_on_demand function in app.py to use these returned ID lists for precise UPDATE statements to set qb_..._sync_status = 'synced' and qb_..._synced_at.
    - **Action:** Implement logic to set status to 'error' and populate qb_..._last_error if a specific record fails IIF generation (e.g., critical mapping failure) or if email sending for its batch fails.
    - **Files:** app.py, iif_generator.py.

4. **Enhance Frontend QuickbooksSync.jsx:**
    - **Action:** Fetch and display actual last sync timestamp from a new backend endpoint (e.g., /api/quickbooks/sync-status-summary).
    - **Action:** Fetch and display counts of pending POs and Sales Orders from a new backend endpoint.
    - **Action:** Improve error message display and user feedback based on the detailed response from /api/quickbooks/trigger-sync.
    - **Files:** QuickbooksSync.jsx, app.py.

- **Phase 1: FedEx Integration Finalization (Status: Blocked/Pending External Factors)**
    1. **FedEx "Bill RECIPIENT" Sandbox Testing (Critical, Blocked):**
        - **Action:** Engage FedEx Developer Support to obtain valid sandbox recipient account numbers or procedures for testing third-party billing.
        - **Goal:** Successfully generate a sandbox FedEx label using paymentType: RECIPIENT.
        - **Files:** shipping_service.py.
    2. **FedEx Production Label Certification (Critical, Pending):**
        - **Action:** Follow FedEx's "Shipping Label Certification steps." Submit sample labels and Label Cover Sheet.
        - **Goal:** Receive approval from FedEx Label Analysis Group.
        - **Files:** shipping_service.py.
    3. **Integrate FedEx Label Generation into app.py/process_order:** (Dependent on above)
        - **Action:** Add logic to call shipping_service.generate_fedex_label when FedEx is chosen. Handle PDF bytes for GCS/email. Update shipments table.
        - **Files:** app.py, shipping_service.py.
    4. **(Optional) Standalone FedEx Label Generator UI & Backend:** (If required)

5. **Production "Bill Recipient" Testing with Real Customer Accounts (Pilot):** (Post-certification)

- **Phase 3: UI/UX Enhancements & Remaining Fixes (Consolidated and re-prioritized)**
    1. **Supplier Management UI (CRUD):** Full robustness check (links, forms, errors, display).
    2. **HPE Description Management UI (CRUD):** Final polish (validations, user feedback).
    3. **Frontend - Eloquia Font Issue (from v11.0):** Resolve font rendering on deployed site.
    4. **Loading Indicators and UX Refinements (General):** Audit API-calling components. Review UX of G1 Onsite, Customer-Billed Shipments, Standalone Label Gen, new QuickBooks Sync UI. Improve error message consistency.
- **Phase 4: MVP Task Completion (Consolidated from previous reports)**
    1. **PO Export Feature (Excel - Backend & Frontend):** Implement GET /api/exports/pos in app.py (using openpyxl). Add UI trigger.
    2. **Finalize Authentication & Authorization Robustness:** Consistent 401/403 handling. Review @verify_firebase_token. Document Custom Claims process if no Admin UI.
- **Phase 5: Infrastructure & Deployment Finalization**
    1. **Postmark - External Domain Sending Approval:** Ensure DKIM, SPF, Custom Return-Path are fully configured for production domain.
    2. **Cloudflare DNS for Production Domains (If applicable).**
- **Phase 6: Long-Term Improvements & Best Practices**
    1. **Backend Daily Revenue Aggregation Timezone:** Refactor /api/reports/daily-revenue if local timezone reporting (e.g., CDT) is needed.
    2. **Security Hardening:** Consider Cloud Run Ingress (IAP). Comprehensive server-side input validation. Enhance structured logging.

3. **Asynchronous Task Processing (e.g., Google Cloud Tasks):** For long-running operations like process_order or batch IIF generation if they become too slow for synchronous API calls.
4. **Admin User Approval UI:** Interface for managing Firebase custom claims (isApproved).
5. **QuickBooks Integration - QBWC Exploration:** If IIF limitations become too problematic (especially for Sales Orders with complex inventory, payments, or need for two-way sync), evaluate and potentially implement QuickBooks Web Connector (QBWC) with qbXML. This would be a significant new development effort.
6. **Data Integrity for QuickBooks Lists (TERMS and SHIPVIA for Sales Orders):** Ensure precise matching if these become part of Sales Order IIF.
7. **Automated Unit/Integration Tests:** Pytest (backend), Jest/React Testing Library (frontend).
8. **Order Comment Parsing Reliability:** Consider BigCommerce Order Metafields as a more robust alternative to parsing freight details from customer_message.

---

8. Known Issues & Considerations (Updated & Consolidated)

- **FedEx "Bill RECIPIENT" Sandbox Testing (ACTIVE - BLOCKED):** Current sandbox tests fail with ACCOUNT.VALIDATION.FAILED. Requires FedEx Developer Support.
- **FedEx Production Label Certification (ACTIVE - PENDING):** Needs completion before full production FedEx shipping is active.
- **QuickBooks Desktop & IIF Limitations:** IIF is fragile, poor error feedback, hard to update QB data. QBWC or QB Online API are more robust future alternatives. (Current IIF work is proceeding as planned).
- **Scalability of Synchronous Processing:** /api/orders/<id>/process and /api/quickbooks/trigger-sync are synchronous. For high volumes, consider asynchronous tasks.
- **Environment Management:** Maintain strict separation for dev, staging (if any), production environments (API keys, DB credentials).
- **Firebase & GCP Quotas:** Monitor usage to avoid hitting quotas.

- **Service Account Key Security (Local Dev):** Protect downloaded Firebase Admin SDK JSON keys.
- **Timezone for Reporting:** Daily Sales Report uses UTC. Backend changes needed for local timezone alignment.
- **BigCommerce API Rate Limits:** Be mindful, especially for bulk operations. Implement retries if needed.

---

9. Summary of Recent Changes (From This Chat Session - May 20, 2025)

- **FedEx Label Testing & Diagnosis:** (Details in Section 2)
- **QuickBooks IIF - Sales Orders & Payments Implementation (iif_generator.py):**
    - Core logic for generating Invoices and corresponding Payments.
    - US Central Time date formatting implemented.
    - State name to 2-letter abbreviation, and comma after city in addresses.
    - Item descriptions now sourced from hpe_description_mappings with fallback.
    - Shipping line description uses actual shipping method (or "UPS Ground" for free shipping) and handles "(ValueB)" extraction.
    - Billing address (from orders table) used for Invoice Bill To fields.
    - Corrected " to "in" in sanitize_field for PO compatibility.
    - Corrected sign convention for Invoice SPL amounts (now negative).
    - Functions now return lists of processed DB IDs.
- **QuickBooks IIF - Backend API & Status Updates (app.py):**
    - ingest_orders updated to fetch and store full BigCommerce billing_address components and shipping_cost_ex_tax into new orders table columns.
    - New /api/quickbooks/trigger-sync endpoint created to call iif_generator for all pending POs and Sales, email IIFs, and update qb_..._sync_status to 'synced' for processed records using returned IDs.

- **QuickBooks IIF - Frontend (QuickbooksSync.jsx):**
    - Component created with a button to trigger /api/quickbooks/trigger-sync.
    - CSS aligned with FormCommon.css.
    - Resolved response.json is not a function error.
- **General Debugging & Refinements:**
    - Addressed various Python errors (indentation, undefined variables, incorrect function arguments, return tuple mismatches) in app.py and iif_generator.py.
    - Corrected orders.status filter for Sales IIF generation to use 'Completed Offline' OR 'Processed' (then changed to remove local status filter entirely for "all pending" mode based on user request).
    - Refined packing slip payment method display in document_generator.py.

---

10. Conclusion for Handover & Immediate Next Steps for Human Developer

The G1 PO App has advanced significantly, with core order processing, document generation, shipping label features (UPS functional, FedEx groundwork laid), and a newly developed QuickBooks Desktop IIF integration module for both Purchase Orders and Sales Orders/Payments. The IIF generation logic is in place, along with a backend API and frontend UI to trigger it.

Immediate Priorities for the Incoming Developer:

1. **Thoroughly Test QuickBooks IIF Import:**
    - Manually import the IIF files generated by iif_generator.py (for both POs and Sales) into a **TEST QuickBooks Desktop company file.**
    - Verify all data points: customer ("G1.com Website Customer" for sales), dates (US Central Time), document numbers, terms, Bill To/Ship To addresses (including new formatting), line items (SKU/Item Name from mappings, descriptions, quantities, prices, amounts with correct signs), shipping lines, tax lines, and payment application.
    - This is the most critical validation step for the new IIF module. Document any discrepancies or import errors.
2. **Verify & Refine Database Status Updates:**

- Confirm that the /api/quickbooks/trigger-sync endpoint in app.py correctly updates the qb_po_sync_status and qb_sales_order_sync_status fields to 'synced' (and qb_..._synced_at) ONLY for the records that were successfully included in an IIF batch and for which the email was (notionally) sent.
- Ensure that subsequent runs of the sync process do not pick up these 'synced' records, thus preventing duplicates.

3. **Enhance QuickbooksSync.jsx Frontend:**
    - Implement fetching and displaying the "Last Successful Sync" timestamp (this will require a new, simple backend endpoint).
    - Implement fetching and displaying "Number of Pending POs" and "Number of Pending Sales Orders" (also requires a new backend endpoint to query counts based on qb_..._sync_status).
    - Ensure robust error display and user feedback.
4. **Address FedEx Blockers:**
    - Follow up with FedEx Developer Support regarding "Bill RECIPIENT" sandbox testing.
    - Drive the FedEx Production Label Certification process to completion. This is essential for live FedEx shipping.

This detailed report, combined with the codebase and previous handover notes, should provide a solid foundation for the incoming developer to effectively take over the project and guide it to completion. The QuickBooks IIF integration is a major step forward and requires careful end-to-end testing.