G1 PO App - Consolidated Handover Report & Project Plan (v13.0)

Date: May 19, 2025 (Reflecting all progress and chat sessions ending May 19, 2025, CDT)

Version: 13.0 (This document supersedes v11.0 and v12.0, and incorporates all subsequent development, troubleshooting, 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.

Table of Contents

1. Project Goal & Core Value Proposition
2. Current State of the Application (As of May 19, 2025)
    - Key Achievements & Functionalities (Consolidated & Updated)
    - Recent Developments & Enhancements (Post v12.0 Report - from this chat session)
3. Technology Stack & Project Layout
    - Backend (order-processing-app directory)
    - Frontend (order-processing-app-frontend directory)
    - Database (PostgreSQL on Google Cloud SQL)
    - Cloud Platform (GCP)
    - Authentication (Firebase Authentication)
    - External APIs
4. Key Code Files & Their Roles
    - Backend
    - Frontend
5. Database Schema Notes & Key Tables
6. Developer Setup & Environment

    - Backend
    - Frontend
    - Running Locally
    - Deployment
    - Crucial Environment Variables
7. Detailed Plan for Completion & Future Tasks
    - Phase 0: Critical Immediate Issues (Post-Handover)
    - Phase 1: FedEx Integration Finalization
    - Phase 2: QuickBooks Integration Module
    - Phase 3: UI/UX Enhancements & Remaining Fixes
    - Phase 4: MVP Task Completion
    - Phase 5: Infrastructure & Deployment Finalization
    - Phase 6: Long-Term Improvements & Best Practices
8. Known Issues & Considerations (Updated & Consolidated)
9. Summary of Recent Changes (From This Chat Session)
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 and G1-stocked items.

Core Functionality:

- Ingest orders from BigCommerce.
- Allow authorized users to manage these orders.
- Generate necessary documentation (POs, Packing Slips, Shipping Labels for UPS & FedEx).
- Facilitate communication with suppliers.
- Update BigCommerce with shipment details and order statuses.

- Integrate with QuickBooks Desktop for accounting (Purchase Orders, with Sales Order sync planned).

Recent Enhancements & Goals (Consolidated):

- **"G1 Onsite Fulfillment":** Process orders fulfilled directly from G1 stock.
- **"Customer-Billed Shipments":** Handle orders where customers use their own UPS or FedEx accounts.
- **"Standalone UPS Label Generator":** Utility to generate UPS labels ad-hoc.
- **UI Reorganization:** Introduction of a "Utilities" section for administrative tools (Supplier Management, HPE Description Management, Standalone Label Generator, QuickBooks Sync placeholder).
- **Dark Mode UI Consistency:** Implemented for form components.
- **Replace T-HUB:** A key goal is to replace the functionality of external tools like T-HUB for BigCommerce-to-QuickBooks order synchronization.

Primary Aim: Reduce manual effort, improve accuracy, provide a centralized platform for critical business workflows, and enhance operational efficiency.

2. Current State of the Application (As of May 19, 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.

Key Achievements & Functionalities (Consolidated & Updated)

- **Order Ingestion & Management:**
    - Successfully ingests relevant orders from BigCommerce (filtered by status, e.g., "Awaiting Fulfillment").
    - Parses BigCommerce customer notes for UPS and FedEx third-party billing details, storing them in the database.
    - Preserves manually set statuses in the local DB and strips sensitive patterns from customer notes during ingestion.
    - 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 and prioritizes their selected shipping service.
- **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.
- **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 with part number link clicks for "new" orders.
    - 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.
    - /api/utils/generate_standalone_ups_label route for ad-hoc UPS label generation.
- **Document Generation (**document_generator.py **- ReportLab):**
    - **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.
- **Shipping Label Generation (**shipping_service.py**):**
    - **UPS:**
        - Fully functional for "Bill Sender" (using older PaymentInformation structure for account EW1847) and "Bill Third Party" (customer account, using current PaymentDetails structure) via OAuth 2.0.
        - Generates PDF labels (via GIF conversion). Label PDF now has 1-inch margins and is top-aligned.
        - *Outstanding Issue:* Intermittently, the label image bytes are not returned from UPS even when a tracking number is, leading to missing label attachments in emails. Enhanced logging is in place to capture the full UPS success response.
    - **FedEx:**
        - 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).

        - "Bill RECIPIENT" sandbox testing for third-party accounts results in ACCOUNT.VALIDATION.FAILED, awaiting FedEx guidance/test accounts.

- **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 Fulfillment notifications and Standalone UPS Labels to sales@globalonetechnology.com.
    - Sends daily IIF batch emails.
- **Cloud Storage (**app.py **- Google Cloud Storage):**
    - Uploads generated POs, Packing Slips, and Labels to GCS.
    - *Outstanding Issue:* Signed URL generation fails with "you need a private key to sign credentials," likely due to missing "Service Account Token Creator" role on the Cloud Run service account itself.
- **BigCommerce Integration (**shipping_service.py**,** app.py**):**
    - Retrieves order data.
    - Refined logic for shipment creation and final order status updates using separate helper functions.
- **QuickBooks Integration (**iif_generator.py**,** app.py**):**
    - Generates and emails a daily IIF file for Purchase Orders (yesterday's via scheduler, today's via user trigger).
- **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 @verify_firebase_token decorator (handles OPTIONS requests for CORS).
    - AuthContext.jsx provides apiService for authenticated frontend calls, used in OrderDetail.jsx, SupplierForm.jsx, EditSupplierForm.jsx, etc.

- **Frontend UI Structure & Utilities (**App.jsx**,** UtilitiesLandingPage.jsx**, etc.):**
    - Navigation: "Orders | Daily Sales | Utilities".
    - **Utilities Landing Page (**UtilitiesLandingPage.jsx**):** Links to:
        - Standalone UPS Label Generator (functional).
        - QuickBooks Integration (placeholder QuickbooksSync.jsx component and route defined).
        - Supplier Management (SupplierList.jsx, SupplierForm.jsx, EditSupplierForm.jsx - routing and apiService integration fixed).
        - HPE Description Management (functional CRUD for hpe_description_mappings with pagination and filtering).
    - **Dark Mode CSS:** Implemented for form components using FormCommon.css and CSS variables.
- **CORS Issues:** Resolved by correcting frontend URL construction and updating verify_firebase_token decorator.

Recent Developments & Enhancements (Post v12.0 Report - From This Chat Session)

- **UPS "Bill Sender" Label Fix:** Resolved the "single billing option required" error by conditionally using the older PaymentInformation payload structure for "Bill Sender" shipments to account EW1847, while "Bill Third Party" uses the current PaymentDetails structure.
- **UPS Label PDF Margins:** Adjusted convert_image_bytes_to_pdf_bytes in shipping_service.py to render the label with 1-inch margins, top-aligned on the PDF.
- **UPS Label Debugging:** Added enhanced logging to shipping_service.py to capture the full JSON response from UPS when a tracking number is obtained, to help diagnose missing label images.
- OrderDetail.jsx **Input Lag:** Fixed by removing purchaseItemsRef and using the purchaseItems state directly for rendering and updates, ensuring responsive input fields.
- **Supplier Management UI Routing:**
    - Corrected "Page not found" errors for /suppliers/add and edit supplier pages by uncommenting and properly defining routes in App.jsx (now pointing to /utils/suppliers/add and /utils/suppliers/edit/:supplierId).

    - Updated SupplierList.jsx links and navigation to align with these routes.
    - Updated UtilitiesLandingPage.jsx link for Supplier Management to /utils/suppliers.
- **Supplier Forms** apiService **Integration:**
    - SupplierForm.jsx and EditSupplierForm.jsx updated to use apiService from AuthContext for backend calls, consistent with other components.
- **CSS Dark Mode Consistency:**
    - Updated FormCommon.css with comprehensive CSS variables for light and dark modes.
    - Modified EditSupplierForm.css to import and leverage FormCommon.css, ensuring consistent dark mode theming for supplier forms.

3. Technology Stack & Project Layout

(Largely consolidated from v11.0 and v12.0, with minor updates)

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)
- **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).
- **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.
- **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.
- **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

(Highlighting recent changes and areas of focus)

Backend

- 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 for UPS/FedEx third-party billing.
    - Main order processing logic in /api/orders/<order_id>/process, orchestrating document generation, label creation, GCS uploads, database updates, and email notifications. Differentiates G1 Onsite, Single Supplier, and Multi-Supplier POs.
    - Supplier CRUD endpoints (/api/suppliers).
    - HPE Description Mapping CRUD endpoints (/api/hpe-descriptions).
    - Standalone UPS label generation (/api/utils/generate_standalone_ups_label).
    - IIF generation trigger routes (/api/tasks/...).
    - Protected by @verify_firebase_token decorator.
- shipping_service.py**:**
    - Manages UPS and FedEx API interactions (OAuth, label generation).
    - **UPS:**
        - generate_ups_label_raw(): Contains the conditional payment structure logic (uses older PaymentInformation for "Bill Sender" EW1847, current PaymentDetails for "Bill Third Party").

        - Includes enhanced logging to capture the full UPS JSON response when a tracking number is obtained, to debug missing label images.
    - **FedEx:**
        - get_fedex_oauth_token(): Retrieves OAuth token (CXS-TP scope working).
        - generate_fedex_label_raw(): Drafted for label generation; "Bill SENDER" tested in production (test-marked label), "Bill RECIPIENT" sandbox testing pending account validation.
    - convert_image_bytes_to_pdf_bytes(): Converts GIF label images to PDF, now with 1-inch margins and top-alignment.
    - Handles BigCommerce shipment creation and order status updates.
- email_service.py**:**
    - Manages sending emails via Postmark API.
    - send_po_email(): Sends PO, Packing Slip, and (if available) Label to supplier. Expects base64 encoded content for attachments from app.py.
    - send_iif_batch_email(): Sends daily IIF batches.
    - send_sales_notification_email(): Used for G1 Onsite and Standalone Label notifications.
- document_generator.py**:**
    - Generates Purchase Order and Packing Slip PDFs using ReportLab.
    - Handles logo fetching from GCS.
    - generate_packing_slip_pdf() accepts is_g1_onsite_fulfillment flag for content variations.
- iif_generator.py**:**
    - Generates IIF content for Purchase Orders for QuickBooks Desktop import.
    - create_and_email_daily_iif_batch(): For yesterday's POs (scheduler).
    - create_and_email_iif_for_today(): For today's POs (user-triggered).

Frontend

- App.jsx**:**
    - Manages all client-side routing using react-router-dom.
    - Includes main navigation structure.
    - Defines ProtectedRoute for authenticated routes.
    - **Recently Updated:** Routes for /utils/suppliers/add and /utils/suppliers/edit/:supplierId are now active and correctly point to their respective components.
- AuthContext.jsx**:**
    - Manages Firebase authentication state (currentUser, loading, logout).
    - Provides apiService (a wrapper around fetch) that automatically attaches the Firebase ID token to backend requests and handles basic response/error parsing.
- OrderDetail.jsx**:**
    - Displays detailed order information and provides forms for processing orders (Single PO, Multi-PO, G1 Onsite).
    - **Recently Fixed:** Input lag in PO item entry fields resolved by using purchaseItems state directly for rendering.
    - Dynamically adapts UI based on selected fulfillment mode.
    - Handles Brokerbin link generation and "RFQ Sent" status updates.
- UtilitiesLandingPage.jsx**:**
    - Provides a dashboard-like interface for accessing various utility modules.
    - **Recently Updated:** Link to Supplier Management now correctly points to /utils/suppliers.
- **Supplier Management Components:**
    - SupplierList.jsx**:** Displays a list of suppliers. Links to add/edit.
        - **Recently Updated:** Uses apiService for fetching suppliers. Links point to correct /utils/suppliers/... paths.
    - SupplierForm.jsx **(Add):** Form for creating new suppliers.

        - **Recently Updated:** Uses apiService.post(). Navigation path corrected.
    - EditSupplierForm.jsx**:** Form for editing existing suppliers.
        - **Recently Updated:** Uses apiService.get() to fetch and apiService.put() to update. Navigation path corrected.
- **HPE Description Management Components:**
    - HpeDescriptionList.jsx, HpeDescriptionForm.jsx, EditHpeDescriptionForm.jsx: Functional CRUD for hpe_description_mappings table, including pagination and filtering.
- **Styling:**
    - FormCommon.css**:** Centralized CSS for forms.
        - **Recently Updated:** Now includes comprehensive CSS variables and @media (prefers-color-scheme: dark) rules for dark mode theming.
    - SupplierForm.css**:** Imports FormCommon.css.
    - EditSupplierForm.css**:**
        - **Recently Updated:** Now imports FormCommon.css to inherit common styles and dark mode support, with redundant styles removed.

5. Database Schema Notes & Key Tables

(Consolidated from v11.0 and v12.0, with updates)

- orders **Table:**
    - Stores ingested BigCommerce order details.
    - Key columns: id, bigcommerce_order_id, customer_name, customer_company, shipping address fields, 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 (VARCHAR): Stores UPS service name if customer-billed.

        - customer_ups_account_number (VARCHAR)
        - is_bill_to_customer_account (BOOLEAN): True if billing to customer's UPS account.
        - customer_ups_account_zipcode (VARCHAR): Customer's UPS account billing ZIP (currently not populated from BigCommerce, may be needed for validation).
    - **FedEx Customer Billing Fields:**
        - customer_selected_fedex_service (VARCHAR)
        - customer_fedex_account_number (VARCHAR)
        - is_bill_to_customer_fedex_account (BOOLEAN)
        - customer_fedex_account_zipcode (VARCHAR): (Currently not populated).
    - **QuickBooks Sync Status (To Be Added - Phase 2):**
        - qb_sales_order_sync_status (VARCHAR: 'pending_sync', 'synced', 'error', 'skipped')
        - qb_sales_order_synced_at (TIMESTAMP)
        - qb_sales_order_last_error (TEXT)
- order_line_items **Table:**
    - Stores individual line items for each order.
    - Links to orders table via order_id.
    - Key columns: id, order_id, bigcommerce_line_item_id, sku (original SKU from BC), name, quantity, sale_price.
- suppliers **Table:**
    - Stores supplier information.
    - Key columns: id, name, email, contact_person, address fields, payment_terms, defaultponotes.
- purchase_orders **Table:**
    - Stores generated Purchase Orders.

    - Links to orders and suppliers tables.
    - 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 (To Be Added - Phase 2):**
        - qb_po_sync_status (VARCHAR)
        - qb_po_synced_at (TIMESTAMP)
        - qb_po_last_error (TEXT)
- po_line_items **Table:**
    - Stores line items for each Purchase Order.
    - Links to purchase_orders and order_line_items (via original_order_line_item_id).
    - Key columns: id, purchase_order_id, original_order_line_item_id, sku (SKU used on PO), description, quantity, unit_cost, condition.
- shipments **Table:**
    - Stores shipment details (tracking, label paths).
    - Links to orders and purchase_orders.
    - purchase_order_id is NULLABLE (for G1 Onsite Fulfillment shipments).
    - Key columns: id, order_id, purchase_order_id, tracking_number, shipping_method_name, weight_lbs, label_gcs_path, packing_slip_gcs_path.
- hpe_part_mappings **Table:**
    - Maps original SKUs to HPE Option PNs and Spare PNs.
    - Columns: sku (original/spare), option_pn, pn_type ('option' or 'spare').
- hpe_description_mappings **Table:**
    - Stores custom PO descriptions for HPE Option PNs. Actively managed via UI.
    - Columns: option_pn (PRIMARY KEY), po_description.
- qb_product_mapping **Table:**

  - (Schema details from previous reports - for mapping app products to QuickBooks items).
- users **Table (Implicit via Firebase):**
  - User authentication is handled by Firebase Authentication. User approval (isApproved custom claim) is managed in Firebase.

6. Developer Setup & Environment

(Consolidated and updated)

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).
3. **Dependencies:** Install using pip install -r requirements.txt.
4. **Environment Variables (**.env **file):**
   - Create a .env file in the 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 (your EW1847 account), UPS_API_ENVIRONMENT (production or test), UPS_API_VERSION.
   - **FedEx API:**
     - Sandbox: FEDEX_API_KEY_SANDBOX, FEDEX_SECRET_KEY_SANDBOX, FEDEX_ACCOUNT_NUMBER_SANDBOX, FEDEX_OAUTH_URL_SANDBOX, FEDEX_SHIP_API_URL_SANDBOX.
     - Production: FEDEX_API_KEY_PRODUCTION, FEDEX_SECRET_KEY_PRODUCTION, FEDEX_ACCOUNT_NUMBER_PRODUCTION,

        FEDEX_OAUTH_URL_PRODUCTION, FEDEX_SHIP_API_URL_PRODUCTION.

        - Common: FEDEX_API_ENVIRONMENT (sandbox or production), FEDEX_GRANT_TYPE (e.g., client_credentials).

    - **Postmark:** EMAIL_API_KEY (Server API Token), EMAIL_SENDER_ADDRESS, EMAIL_BCC_ADDRESS, QUICKBOOKS_EMAIL_RECIPIENT.
    - **GCS:** GCS_BUCKET_NAME, COMPANY_LOGO_GCS_URI.
    - **Ship From Address:** SHIP_FROM_NAME, SHIP_FROM_CONTACT, SHIP_FROM_STREET1, SHIP_FROM_STREET2, SHIP_FROM_CITY, SHIP_FROM_STATE, SHIP_FROM_ZIP, SHIP_FROM_COUNTRY, SHIP_FROM_PHONE.
    - **Firebase/GCP:** GOOGLE_APPLICATION_CREDENTIALS (path to your Firebase Admin SDK service account key JSON file for local development, associated with project g1-po-app-77790).
    - **Flask:** FLASK_DEBUG (True or False).

5. **Google Cloud SQL Auth Proxy:** Required for connecting to the Cloud SQL PostgreSQL database locally. Download and run it, configured for your Cloud SQL instance.

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

1. **Node.js:** LTS version recommended.
2. **Dependencies:** Install using 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, VITE_FIREBASE_PROJECT_ID, VITE_FIREBASE_STORAGE_BUCKET, VITE_FIREBASE_MESSAGING_SENDER_ID, VITE_FIREBASE_APP_ID. (Get these from your Firebase project settings).

Running Locally

1. **Start Backend:**

  - Ensure Cloud SQL Auth Proxy is running.
  - In the order-processing-app directory: flask run --port=8080 (or your configured port, e.g., python app.py).
2. **Start Frontend:**
  - In the order-processing-app-frontend directory: npm run dev.

Deployment

- **Backend (Cloud Run):**
  - Dockerized, pushed to Google Artifact Registry, deployed to Cloud Run.
  - **Cloud Run Service Account IAM Roles (CRITICAL):**
    - "Cloud SQL Client" (for DB connection).
    - "Secret Manager Secret Accessor" (to access secrets).
    - "Storage Object Admin" (or more granular read/write to the specific GCS bucket).
    - **"Service Account Token Creator" granted TO ITSELF** (for GCS V4 Signed URLs - currently an outstanding issue).
    - Any other roles needed for services it interacts with (e.g., Pub/Sub if Cloud Tasks are used via Pub/Sub).
- **Frontend (Firebase Hosting):**
  - Build with npm run build.
  - Deploy using Firebase CLI: firebase deploy --only hosting.

7. Detailed Plan for Completion & Future Tasks

(Consolidated, re-prioritized, and incorporating recent developments)

Phase 0: Critical Immediate Issues (Post-Handover)

1. **Resolve UPS Label Image Missing in Email (HIGH PRIORITY):**
  - **Symptom:** Tracking number generated and sent to BigCommerce, but the label PDF is not attached to the supplier email because label_image_base64 is None in shipping_service.py.
  - **Action:**

        - Deploy the latest shipping_service.py (ID: shipping_service_py_final_handover) which includes logging of the *full successful JSON response* from UPS.
        - Trigger the issue and analyze the logged DEBUG UPS_LABEL_RAW (Full Response for Success Code 1): ... message in Cloud Logging. Examine the ShipmentResults.PackageResults[0].ShippingLabel.GraphicImage path in the response.
        - **Verify UPS Account** EW1847**:**
            - Confirm with UPS that the account is fully active and authorized for API shipping requests, including image retrieval.
            - Confirm there are no service-level restrictions that might prevent label image generation for certain services/destinations.
        - **Verify** SHIP_FROM_... **Address:** Meticulously compare all SHIP_FROM_... environment variables against the exact address details registered with UPS for account EW1847. Any mismatch can cause validation issues.
        - **Consider UPS API Label Format:** If GIF continues to be problematic and the full response shows no GraphicImage, investigate if UPS can return PDF directly for the services used. This would require adjusting convert_image_bytes_to_pdf_bytes to handle a direct PDF passthrough or be bypassed for UPS if it returns PDF.
    - **File(s) Involved:** shipping_service.py, GCP IAM, UPS Account Settings.

2. **Resolve GCS Signed URL Generation Error (HIGH PRIORITY):**
    - **Symptom:** Logs show "you need a private key to sign credentials. the credentials you are currently using <class 'google.auth.compute_engine'>..."
    - **Action:**
        - In GCP IAM, ensure the Cloud Run service account (e.g., your-project-id@appspot.gserviceaccount.com or a custom one) has the **"Service Account Token Creator"** role granted **to itself**.

        - If using default credentials and impersonation was intended, review the generate_signed_url calls in app.py (specifically in the process_order route within the storage_client block). Ensure service_account_email and access_token=None are correctly used if specifying the service account to do the signing.
        - After any IAM changes, redeploy the Cloud Run service and allow time for permissions to propagate.
    - **File(s) Involved:** app.py, GCP IAM.

Phase 1: FedEx Integration Finalization

(Carry-over from v12.0, dependent on Phase 0 resolutions for general stability)

1. **FedEx "Bill RECIPIENT" Sandbox Testing (Critical):**
    - **Action:** Engage FedEx Developer Support to obtain valid sandbox recipient account numbers or procedures for testing third-party billing where the recipient account is different from the shipper account (e.g., for account 740561073 if that's the test shipper).
    - **Goal:** Successfully generate a sandbox FedEx label using paymentType: RECIPIENT with a true third-party test account.
    - **File(s) Involved:** shipping_service.py (testing block, generate_fedex_label_raw).
2. **FedEx Production Label Certification (Critical):**
    - **Action:** Follow FedEx's "Shipping Label Certification steps." Generate sample sandbox labels (PDF/ZPL) for all services intended for production use. Submit the Label Cover Sheet with PRODUCTION API Key and Account Number.
    - **Goal:** Receive approval from FedEx Label Analysis Group to activate production credentials for live shipping.
    - **File(s) Involved:** shipping_service.py.
3. **Integrate FedEx Label Generation into** app.py/process_order**:**
    - **Action:** Once certified and "Bill Recipient" is working, add logic to app.py in the process_order route to call shipping_service.generate_fedex_label when FedEx is the chosen carrier (similar to how UPS labels are handled).

    - Ensure correct data (order details, ship from, weight, method) is passed.
    - Handle FedEx label PDF bytes for GCS upload and email attachment.
    - Update database shipments table with FedEx tracking and label path.
    - **File(s) Involved:** app.py, shipping_service.py.

4. **(Optional) Standalone FedEx Label Generator UI & Backend:**
    - **Action:** If required, create a new frontend component similar to StandaloneUpsLabel.jsx and a new backend route in app.py (e.g., /api/utils/generate_standalone_fedex_label) to generate ad-hoc FedEx labels.
    - **File(s) Involved:** New .jsx component, App.jsx (routing), app.py.
5. **Production "Bill Recipient" Testing with Real Customer Accounts (Pilot):**
    - **Action:** After production certification, conduct pilot tests with consenting customers for FedEx "Bill Recipient" shipments.
    - **File(s) Involved:** Full stack.

Phase 2: QuickBooks Integration Module (New Major Feature)

(As detailed in v12.0 - Goal: Replicate T-HUB BigCommerce-to-QuickBooks sales order sync, add on-demand PO/Sales Order sync, track sync status. Assumed QuickBooks Desktop & IIF files.)

1. **Task 1: Finalize QuickBooks Version & Integration Strategy Detail:**
    - **Action:** Confirm definitively if QuickBooks Desktop or Online is the target. If Desktop, re-evaluate if IIF is sufficient for sales orders vs. QBWC. (Plan assumes IIF for Desktop).
2. **Task 2: Database Schema for QuickBooks Sync Status:**
    - **Action:** Add qb_sales_order_sync_status, qb_sales_order_synced_at, qb_sales_order_last_error to orders table.
    - **Action:** Add qb_po_sync_status, qb_po_synced_at, qb_po_last_error to purchase_orders table.
    - Default qb_..._sync_status to 'pending_sync'.
3. **Task 3: Backend Logic for Sales Order Data Extraction (from** orders **table):**

- **Action:** Create function in app.py or quickbooks_service.py to fetch eligible orders (status 'Completed Offline'/'Shipped' AND qb_sales_order_sync_status is 'pending_sync'/'error').
- Map orders and order_line_items data to fields for QB Sales Receipt/Invoice (Customer, Item, Qty, Rate, Amount, Tax, Shipping, SO#, Date).

4. **Task 4: Backend Logic for Purchase Order Data Extraction (from** purchase_orders **table):**
    - **Action:** Similar to Task 3, but for purchase_orders where qb_po_sync_status is 'pending_sync'/'error'.
5. **Task 5: IIF File Generation for Sales Orders:**
    - **Action:** Extend iif_generator.py to generate IIF for Sales Receipts/Invoices. Handle customer mapping and item mapping (ensure names/SKUs match QB).
6. **Task 6: Modify IIF File Generation for Purchase Orders (if needed):**
    - **Action:** Review existing PO IIF generation in iif_generator.py. Ensure alignment with on-demand sync and status tracking.
7. **Task 7: Backend API Endpoint for On-Demand QuickBooks Sync:**
    - **Action:** Create POST /api/quickbooks/trigger-sync in app.py.
    - This endpoint will call data extraction, IIF generation, email IIF file(s), and update qb_..._sync_status and qb_..._synced_at in the database. Log errors to qb_..._last_error.
8. **Task 8: Frontend UI for QuickBooks Sync (**QuickbooksSync.jsx**):**
    - **Action:** Develop UI with "Sync Now" button, display for last sync status, errors, counts of pending items. Button calls backend endpoint.
9. **Task 9: Thorough Testing of QuickBooks Sync:**
    - **Action:** Test with various scenarios, verify IIF import into a TEST QuickBooks company file. Check data accuracy, transaction types, customer/item creation. Test error handling.

Phase 3: UI/UX Enhancements & Remaining Fixes

(Consolidated and re-prioritized)

1. **Supplier Management UI (CRUD):**
   - **Action:** Ensure SupplierList.jsx, SupplierForm.jsx, and EditSupplierForm.jsx are fully robust. Verify all links, form submissions, error handling, and data display.
   - **File(s) Involved:** SupplierList.jsx, SupplierForm.jsx, EditSupplierForm.jsx, App.jsx.
2. **HPE Description Management UI (CRUD):**
   - **Action:** Final polish on HpeDescriptionForm.jsx and EditHpeDescriptionForm.jsx (e.g., input validations, user feedback).
   - **File(s) Involved:** HpeDescriptionList.jsx, HpeDescriptionForm.jsx, EditHpeDescriptionForm.jsx.
3. **Frontend - Eloquia Font Issue (from v11.0):**
   - **Action:** Resolve font rendering on the deployed site. Check @font-face in App.css (or global CSS), font file paths in public folder, and build process.
4. **Loading Indicators and UX Refinements (General):**
   - **Action:** Audit API-calling components for consistent and clear loading indicators.
   - Review UX of G1 Onsite Fulfillment, Customer-Billed Shipments, Standalone Label Generator, and new QuickBooks Sync UI for clarity and ease of use.
   - Improve error message display consistency across the application.

Phase 4: MVP Task Completion

(Consolidated from previous reports)

1. **PO Export Feature (Excel - Backend & Frontend):**
   - **Action (Backend):** Implement GET /api/exports/pos in app.py using a library like openpyxl to generate an Excel file of purchase orders (potentially with filters for date range, supplier).
   - **Action (Frontend):** Add UI elements (e.g., in Dashboard.jsx or a new report section) to trigger this export and download the file.
2. **Finalize Authentication & Authorization Robustness:**

- **Action (Frontend):** Ensure consistent handling of 401/403 errors from apiService, potentially redirecting to login or showing clear messages.
- **Action (Backend):** Review all protected routes and ensure @verify_firebase_token is applied correctly.
- **Action (Documentation):** Document the process for setting Firebase Custom Claims (e.g., isApproved) for new users if an Admin UI is not yet built.

Phase 5: Infrastructure & Deployment Finalization

(Consolidated from previous reports)

1. **Postmark - External Domain Sending Approval:**
   - **Action:** Ensure Postmark is fully configured (DKIM, SPF, Custom Return-Path) for the production domain to maximize email deliverability and avoid spam filters.
2. **Cloudflare DNS for Production Domains (If applicable):**
   - **Action:** If using custom domains (e.g., g1po.com, api.g1po.com), configure DNS records correctly in Cloudflare (or your DNS provider) to point to Firebase Hosting (frontend) and Cloud Run (backend).

Phase 6: Long-Term Improvements & Best Practices

(Consolidated from previous reports, for future consideration)

1. **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. **Security Hardening:**
   - Cloud Run Ingress (consider IAP - Identity-Aware Proxy for an additional security layer).
   - Comprehensive server-side input validation for all API endpoints.
   - Review and enhance structured, detailed, leveled logging.
3. **Asynchronous Task Processing:** For long-running operations like process_order (document generation, external API calls), consider using Google Cloud Tasks to improve API response times and robustness.

4. **Admin User Approval UI:** Implement an admin interface within the app for managing Firebase custom claims (isApproved), replacing any manual scripts.
5. **Data Integrity for QuickBooks Lists (TERMS and SHIPVIA):** Ensure data used for IIF generation (especially for Sales Orders) precisely matches QuickBooks lists to prevent import errors.
6. **Automated Unit/Integration Tests:** Implement a testing suite using Pytest (backend) and Jest/React Testing Library (frontend) to improve code quality and reduce regressions.
7. **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 error-prone.

8. Known Issues & Considerations (Updated & Consolidated)

- **UPS Label Image Missing in Email (ACTIVE - HIGH PRIORITY):**
    - **Symptom:** Tracking number is generated by UPS and updated in BigCommerce, but the label PDF (derived from image bytes) is not attached to the supplier email. Logs indicate label_image_base64 is None in shipping_service.py even on successful UPS API calls that return a tracking number.
    - **Current Diagnostic:** shipping_service.py (ID: shipping_service_py_final_handover) now logs the full successful JSON response from UPS to analyze the PackageResults structure.
    - **Potential Causes:** UPS account restriction for EW1847, incorrect SHIP_FROM_... address validation, service-specific issue with label image return, or unexpected API response structure.
- **GCS Signed URL Generation Error (ACTIVE - HIGH PRIORITY):**
    - **Symptom:** Logs show "you need a private key to sign credentials..." when app.py attempts to generate signed URLs for GCS documents.
    - **Likely Cause:** The Cloud Run service account lacks the "Service Account Token Creator" role granted *to itself*.
- **FedEx "Bill RECIPIENT" Sandbox Testing:**

  - **Status:** Blocked. Current sandbox tests for "Bill RECIPIENT" (third-party) result in ACCOUNT.VALIDATION.FAILED.
  - **Action Needed:** Engage FedEx Developer Support for valid test recipient accounts or clarification on sandbox setup for third-party billing with the shipper account 740561073 (or current test shipper).
- **FedEx Production Label Certification:**
  - **Status:** Pending. The FedEx label evaluation process needs to be completed before production credentials are fully active for live shipping.
- **QuickBooks Desktop & IIF Limitations:**
  - IIF is a fragile format. It doesn't provide good feedback on import errors, cannot easily update existing QB transactions, and handling inventory/payments can be complex. If issues arise during QuickBooks integration (Phase 2), QBWC (QuickBooks Web Connector) or migrating the client to QB Online (with its API) are more robust alternatives.
- **Scalability of Synchronous Processing:** The current /api/orders/<id>/process route is synchronous. For higher volumes, consider moving label/document generation and external API calls to asynchronous tasks (Google Cloud Tasks) to improve API responsiveness.
- **Environment Management:** Maintain strict separation and configuration for development, staging (if any), and production environments, especially for API keys and database credentials.
- **Firebase & GCP Quotas:** Monitor usage of Firebase services (Auth, Hosting) and GCP resources (Cloud Run, Cloud SQL, GCS, Secret Manager, Logging, APIs) to avoid hitting quotas.
- **Service Account Key Security (Local Dev):** Protect any downloaded Firebase Admin SDK service account key JSON files diligently. Use them only for local development.
- **Timezone for Reporting:** The Daily Sales Report currently uses UTC day aggregation. Backend changes may be needed for local business day alignment if required.
- **BigCommerce API Rate Limits:** Be mindful of BigCommerce API rate limits, especially during bulk operations like order ingestion. Implement retries with backoff if necessary.

9. Summary of Recent Changes (From This Chat Session - May 19, 2025)

- **UPS Label Generation (Bill Sender):**
    - Resolved "single billing option required" error by conditionally using the older PaymentInformation payload structure in shipping_service.py for "Bill Sender" shipments (account EW1847). "Bill Third Party" continues to use the current PaymentDetails structure.
    - Adjusted PDF generation in shipping_service.py to render UPS labels with 1-inch margins and top-alignment.
    - Added enhanced logging to shipping_service.py to capture the full UPS JSON response when a tracking number is obtained, to aid debugging the intermittent missing label image issue.
- **Frontend (**OrderDetail.jsx**):**
    - Fixed input lag in PO item entry fields by removing purchaseItemsRef and using the purchaseItems state directly for rendering and updates.
- **Frontend UI Routing & Forms (Supplier Management):**
    - Resolved "Page not found" errors for /suppliers/add and edit supplier pages by defining correct routes in App.jsx (now /utils/suppliers/add, /utils/suppliers/edit/:supplierId).
    - Updated SupplierList.jsx: Links and navigation now use correct paths; API calls switched to apiService.
    - Updated SupplierForm.jsx (Add Supplier): Uses apiService; navigation paths corrected.
    - Updated EditSupplierForm.jsx: Uses apiService; navigation paths corrected.
    - Updated UtilitiesLandingPage.jsx: Link for "Supplier Management" now correctly points to /utils/suppliers.
- **Frontend CSS (Dark Mode):**
    - Updated FormCommon.css with comprehensive CSS variables and @media (prefers-color-scheme: dark) rules for robust dark mode theming across forms.

    - Simplified EditSupplierForm.css to import and leverage FormCommon.css, ensuring consistent dark mode appearance. SupplierForm.css already correctly imported FormCommon.css.

10. Conclusion for Handover

The G1 PO App has reached a significant stage of development, with core functionalities for order processing, document generation (POs, Packing Slips), UPS label generation (including customer-billed and standalone), and initial FedEx integration groundwork in place. The UI has been substantially built out, including a utilities section for administrative tasks, and recent efforts have focused on resolving critical bugs and enhancing UI consistency (dark mode, input responsiveness).

Immediate Priorities for the Incoming Developer:

1. **Resolve UPS Label Image Issue:** This is the most critical operational bug. The label tracking number is generated, but the label image data is not consistently retrieved from UPS, preventing its attachment to supplier emails. The latest shipping_service.py includes enhanced logging to capture the full UPS response, which will be essential for diagnosis. This may involve further investigation into UPS account settings for EW1847 or the specifics of the API response structure. UPDATE: THIS HAS BEEN RESOLVED
2. **Resolve GCS Signed URL Generation Error:** Fix the "private key needed" error by ensuring the Cloud Run service account has the "Service Account Token Creator" role on itself. This is crucial for accessing generated documents. UPDATE: THIS HAS BEEN RESOLVED

Once these critical issues are addressed, the developer should proceed with:

- Finalizing FedEx integration (sandbox testing for "Bill Recipient", production certification, and full integration into the order processing workflow).
- Developing the QuickBooks Integration Module, which is a major new feature request.
- Addressing remaining UI/UX refinements and MVP tasks as outlined in the detailed plan.

The backend API in Flask (Python) and the frontend in React provide a solid foundation. This consolidated report, along with the codebase and previous handover notes, should equip the incoming developer to effectively take over the project and guide it to completion.

Good luck!