G1 PO App - Comprehensive Handover Report & Next Steps

Date: May 15, 2025 (Reflecting current progress and chat session ending May 15, 2025 CDT)

Prepared For: Incoming Developer

Prepared By: Mark (via collaboration with AI Assistant Gemini)

Version: 4.0 (Supersedes report dated May 14, 2025, version 3.0)

1. Introduction

This document provides an updated and comprehensive overview and handover for the "G1 PO App" project. It consolidates information from previous handover reports with all subsequent development progress. This includes:

- Implementation and debugging of the daily IIF file generation for QuickBooks Desktop 2020.
- Critical updates to backend services (app.py, shipping_service.py), including fixes for order ingestion logic (status preservation, note stripping) and authentication for scheduled tasks.
- Successful setup of automated task triggering via Google Cloud Scheduler.
- Integration and extensive troubleshooting of Firebase Authentication (Google Sign-In) for securing the application, including project ID alignment and service account configuration.
- Frontend enhancements for user experience, including dynamic spare part lookups on the Order Detail page, cosmetic improvements to data display, and consistent global button styling.
- Setup of the application's browser page title and favicon.

The goal remains to provide a clear and detailed guide for the incoming developer to understand the project's current state, architecture, codebase, deployment, security model, and the remaining tasks to achieve the Minimum Viable Product (MVP) and future enhancements.

2. Project Overview (Recap)

Goal: To develop a web application ("G1 PO App") that automates the purchase order (PO) and shipment process for drop-shipped items originating from BigCommerce orders, with secure access for authorized users.

Core Functionality (Original Intent & Current Implementation):

- Ingest relevant orders from BigCommerce (filtered by status, e.g., "Awaiting Fulfillment").
- Provide a web interface for authorized users to review these orders.
- Allow authorized users to trigger processing for selected orders.
- Generate Purchase Orders (PDFs) using specific HPE Option PNs mapped from original BigCommerce SKUs.
- Generate Packing Slips (PDFs).
- Generate UPS Shipping Labels (PDFs) via UPS API (OAuth 2.0).
- Email generated PO, Packing Slip, and Label to the selected supplier via Postmark API.
- Upload generated documents (PO, Packing Slip, Label) to Google Cloud Storage (GCS).
- Update order status and add tracking information in BigCommerce via its API.
- Update local application database status for processed orders (orders and purchase_orders tables).
- Provide a web interface (React/Vite) for viewing/filtering orders, viewing details, interactively preparing POs, and triggering processing - secured with user authentication.
- **COMPLETED & VERIFIED:** Generate a daily IIF (Intuit Interchange Format) file for Purchase Orders processed on the previous day, suitable for import into QuickBooks Desktop 2020. This IIF file is emailed to a designated address.
- **COMPLETED & VERIFIED:** User authentication via Firebase (Google Sign-In using project g1-po-app-77790).
- **COMPLETED & VERIFIED (Basic):** Authorization mechanism using Firebase Custom Claims (isApproved: true) to restrict app access.
- **NEW:** Functionality in Order Detail page to look up and display associated spare parts for option SKUs.
- (Future) Manage suppliers and product mappings more extensively via the UI.
- (Future) More robust admin interface for managing user approvals.

3. Current Status (As of May 15, 2025)

The application is largely functional, with core backend processing deployed, IIF generation operational, and a robust user authentication and authorization system implemented and verified. Recent fixes have addressed order ingestion logic, scheduled task authentication, and frontend display details.

3.1. Backend (Python/Flask - app.py and service modules)

- **Deployment:** Dockerized and deployed to Google Cloud Run.
    - Service Name: g1-po-app-backend-service
    - Region: us-central1
    - URL: https://g1-po-app-backend-service-992027428168.us-central1.run.app (Note: Cloud Run URL might change on redeployments if not mapped to a custom domain).
- **Database:** Connected to a PostgreSQL instance on Google Cloud SQL.
    - Instance Connection Name: order-processing-app-458900:us-central1:order-app-db
- **Secrets Management:** All sensitive credentials managed via Google Cloud Secret Manager.
- **Key API Endpoints:**
    - /api/ingest_orders (POST): Functional. Fetches orders from BigCommerce.
        - Logic updated to preserve manually set statuses (e.g., "pending", "Completed Offline", "RFQ Sent") by correcting case sensitivity issues.
        - Logic added to strip ********** text ********** patterns from customer notes before saving.
    - /api/orders, /api/orders/<id> (GET): Functional.
    - /api/orders/<id>/process (POST): Core workflow functional.
    - /api/orders/<id>/status (POST): Functional for manual status updates.
    - /api/orders/status-counts (GET): Functional, provides counts of orders by status for the frontend dashboard.

    - /api/suppliers/**, /api/products/** (GET, POST, PUT, DELETE): CRUD endpoints functional.
    - /api/lookup/description/** (GET): Functional.
    - **NEW:** /api/lookup/spare_part/<option_sku> (GET): Looks up and returns a spare part SKU associated with a given option SKU from the hpe_part_mappings table. Protected by @verify_firebase_token.
    - /api/tasks/trigger-daily-iif (POST): Functional. Triggered by Cloud Scheduler.
        - **FIXED:** Authentication for this endpoint now correctly relies on IAM (Cloud Scheduler's service account with "Cloud Run Invoker" role) by removing the @verify_firebase_token decorator.
- **Authentication & Authorization (Firebase):**
    - Firebase Admin SDK initialized in app.py explicitly targeting Firebase project g1-po-app-77790. This resolved previous "incorrect 'aud' claim" and "auth/quota-exceeded" errors.
    - The Cloud Run service's runtime service account (992027428168-compute@developer.gserviceaccount.com) has been granted appropriate Firebase Admin permissions on the g1-po-app-77790 project.
    - @verify_firebase_token decorator applied to all relevant data-accessing and action-performing API routes (except the IIF task trigger). This decorator validates the 'Bearer' token and checks for the isApproved: true custom claim.
- **CORS:** Configured in app.py to allow requests from the Firebase Hosting frontend URL (https://g1-po-app-77790.web.app) and http://localhost:5173 for local development.
- **Error Handling:** UPS label generation error ("Missing or invalid ship to state province code") was previously addressed.

3.2. Frontend (React/Vite - order-processing-app-frontend)

- **Deployment:** Deployed to Firebase Hosting.
    - URL: https://g1-po-app-77790.web.app
- **Page Title & Favicon:**

    - The browser page title is set to "G1 PO App" (or similar) via the <title> tag in public/index.html.
    - A comprehensive set of favicons (favicon.ico, various PNG sizes, apple-touch-icon.png) are placed in the public directory and linked in public/index.html.
- **Authentication & Authorization (Firebase Client SDK):**
    - Integrated for client-side authentication using Firebase project g1-po-app-77790.
    - Google Sign-In implemented.
    - AuthContext.jsx manages global authentication state.
    - Login.jsx provides the sign-in UI.
    - ProtectedRoute.jsx guards routes.
    - App.jsx includes AuthProvider.
- **Component Updates & Styling:**
    - Global button styling (.btn, .btn-gradient, .btn-shadow-lift, color variants like .btn-primary, .btn-success) defined in src/App.css and applied to buttons in Dashboard.jsx and OrderDetail.jsx for a consistent look and feel (gradient, shadow, lift effects).
    - Dashboard.jsx: "IMPORT NEW ORDERS" button updated to use global styles.
    - OrderDetail.jsx:
        - "PROCESS ORDER" button updated to use global styles, with specific overrides for size.
        - "Mark as Completed Offline" button updated to use global styles.
        - "BACK TO DASHBOARD" button added for "Processed" orders, using global styles.
        - "Or, Process Manually (Set to Pending)" link is now centered when visible. The "Updating..." state of this link also uses the new button styling.
        - **NEW:** Displays associated spare part SKUs (hyperlinked to Brokerbin) next to original SKUs in the "Order Information" card if the original SKU

is an "option" type and has a corresponding spare. This involves fetching data from the new /api/lookup/spare_part/ backend endpoint.

        - **FIXED:** Leading/trailing spaces removed from SKU text within Brokerbin hyperlinks.
        - **FIXED:** Bullet points removed from the displayed list of original SKUs.
        - **FIXED:** "Spare: " prefix text removed from the spare part display.
    - Relevant component-specific CSS files (Dashboard.css, OrderDetail.css) updated to remove redundant styles now handled by App.css and to add specific override classes where needed.
- **Font Issue:** Eloquia font rendering issue on the deployed Firebase site (as noted in previous report) likely still pending.

3.3. Database (PostgreSQL on Cloud SQL)

- Core tables (orders, order_line_items, purchase_orders, po_line_items, suppliers, hpe_part_mappings, hpe_description_mappings, products, qb_product_mapping, shipments) are defined and functional.
- orders.status values are now consistently handled (e.g., "pending" vs. "Pending") due to fixes in app.py.

3.4. Integrations

- **UPS API:** Functional.
- **BigCommerce API:** Functional.
- **Postmark API:** Functional.
- **Google Cloud Storage (GCS):** Functional.
- **Firebase Authentication:** Integrated and working correctly for user sign-in and backend token verification.

3.5. Scheduling (IIF Generation)

- **Google Cloud Scheduler:** Job configured to call /api/tasks/trigger-daily-iif via HTTP POST.

- **Authentication: FIXED.** Now correctly uses OIDC token with the Cloud Scheduler's service account (which has "Cloud Run Invoker" role on the backend service). The IIF generation task should now run successfully.

4. Technology Stack (Recap & Additions)

- **Backend:** Python 3.9+, Flask, SQLAlchemy, pg8000, requests, google-cloud-sql-connector, google-cloud-storage, reportlab, Pillow, Flask-CORS, postmarker, python-dotenv, gunicorn, firebase-admin, **re** (for regular expressions).
- **Frontend:** React, Vite, react-router-dom, axios (if still used, or fetch API), firebase (client SDK).
- **Database:** PostgreSQL (on Google Cloud SQL).
- **Cloud Platform (GCP):** Cloud Run, Cloud SQL, Artifact Registry, Secret Manager, GCS, Cloud Scheduler.
- **Firebase:** Firebase Hosting (for frontend), Firebase Authentication (for user auth - project g1-po-app-77790).
- **APIs:** UPS, BigCommerce, Postmark.

5. Codebase Structure (Key Files & Recent Changes)

5.1. Backend (order-processing-app directory / app.py)

- **app.py:**
    - import re added for regular expression operations.
    - **Firebase Admin SDK Initialization:** Correctly initialized to explicitly target Firebase project g1-po-app-77790.
    - **ingest_orders function:**
        - Logic to strip ********** text ********** patterns from customer_message using re.sub() before saving to the database.
        - Corrected case sensitivity issues when checking statuses_to_preserve (e.g., checking for "pending" instead of "Pending" if DB stores lowercase).
    - **/api/orders/<id> (get_order_details):**

        - Modified to join hpe_part_mappings to fetch hpe_pn_type for the original_sku of each line item. This information is crucial for the frontend to decide whether to perform a spare part lookup.
    - **NEW Endpoint: /api/lookup/spare_part/<option_sku>:**
        - Added to look up and return a spare part SKU associated with a given option SKU (which is assumed to be the original_sku of an 'option' type part from a line item).
    - **/api/tasks/trigger-daily-iif endpoint:**
        - **FIXED:** Removed the @verify_firebase_token decorator. Authentication is now correctly handled by IAM for calls from Cloud Scheduler.
- **requirements.txt:** Includes firebase-admin, google-cloud-sql-connector[pg8000], requests, Flask, SQLAlchemy, etc.
- **shipping_service.py:** State code mapping for UPS labels was previously fixed.
- **iif_generator.py:** Core logic for generating IIF files. Relies on app.engine for database connection when called from app.py.
- **email_service.py:** Handles sending emails via Postmark.
- **document_generator.py:** Handles PDF generation for POs and Packing Slips.
- **setAdminClaim.cjs (External Script):** For manually setting custom claims (like isApproved: true) on Firebase users. Requires Firebase Admin SDK and a service account key JSON file from project g1-po-app-77790. **Store securely, DO NOT COMMIT.**

5.2. Frontend (order-processing-app-frontend directory)

- **public/index.html:**
    - <title> tag updated to "G1 PO App" (or desired application title).
    - <link> tags added for a comprehensive set of favicons (favicon.ico, favicon-16x16.png, favicon-32x32.png, apple-touch-icon.png, android-chrome-192x192.png), all placed in the public/ directory.
- **src/firebase.js (or firebaseConfig.js):** Contains Firebase project configuration for g1-po-app-77790.

- **src/contexts/AuthContext.jsx:** Manages global authentication state.
- **src/components/Login.jsx:** Provides UI for Google Sign-In.
- **src/components/ProtectedRoute.jsx:** Guards routes based on authentication.
- **src/App.jsx:** Main application component, wraps content with AuthProvider.
- **src/App.css:**
    - **NEW:** Added global button styling classes (.btn, .btn-gradient, .btn-shadow-lift) and color variants (.btn-primary, .btn-success) to provide consistent button aesthetics (gradient, shadow, lift effects). Includes adaptive styles for light/dark mode.
- **src/components/Dashboard/Dashboard.jsx:**
    - "IMPORT NEW ORDERS" button className updated to use the new global button styles from App.css (e.g., btn btn-gradient btn-shadow-lift btn-primary).
- **src/components/Dashboard/Dashboard.css:**
    - Removed specific .ingest-button styles, as these are now handled by global classes. Added padding to .dashboard-container to prevent overlap with the sticky ingest button.
- **src/components/OrderDetail/OrderDetail.jsx:**
    - **Spare Part Lookup:**
        - Added state (lineItemSpares, loadingSpares) to manage spare part data.
        - New useEffect hook triggers when orderData.line_items are available. It iterates through line items and, if an item's hpe_pn_type (for its original_sku) is 'option', it calls the new /api/lookup/spare_part/ backend endpoint.
        - Fetched spare SKUs are stored in lineItemSpares.
    - **Display Logic for SKUs (in "Order Information" card):**
        - Renders the original_sku (hyperlinked to Brokerbin).

        - If a corresponding spareSkuForItem exists in lineItemSpares, it's displayed in parentheses next to the original SKU, also hyperlinked to Brokerbin.
        - hpe_option_pn is displayed if it's different from both the original and the spare SKU.
        - **FIXED:** Leading/trailing spaces removed from all SKU text within Brokerbin <a> tags.
        - **FIXED:** Bullet points removed from the SKU list display (changed from div.order-info-item to p.order-info-sku-line).
        - **FIXED:** "Spare: " prefix text removed.
    - **Button Styling:**
        - "PROCESS ORDER" button className updated to btn btn-gradient btn-shadow-lift btn-success process-order-button-specifics.
        - "Mark as Completed Offline" button className updated to btn btn-gradient btn-shadow-lift btn-success manual-action-button-specifics.
        - "BACK TO DASHBOARD" button (for "Processed" orders) className updated to btn btn-gradient btn-shadow-lift btn-primary back-to-dashboard-button-specifics.
        - "Or, Process Manually (Set to Pending)" link:
            - Container div now has textAlign: 'center'.
            - When in "Updating..." state, className includes link-button-updating to apply button-like styling with gradient/shadow/lift.
- **src/components/OrderDetail/OrderDetail.css:**
    - Removed redundant button styles now handled by App.css.
    - Added/updated specific override classes (.process-order-button-specifics, .back-to-dashboard-button-specifics, .manual-action-button-specifics) for styles unique to these buttons (e.g., min-height).
    - Added styles for .order-info-sku-line to ensure proper display without bullets.
    - Added styles for .link-button-updating to match the new button aesthetics.

6. API Integrations & Credentials

- **Backend Secrets:** Managed in Google Cloud Secret Manager for deployed services (DB credentials, BigCommerce tokens, UPS keys, Postmark token, etc.).
- **Local Development (.env files):**
    - Backend: .env file in order-processing-app for all backend secrets.
    - GOOGLE_APPLICATION_CREDENTIALS environment variable should point to the Firebase Admin SDK service account key JSON file (from Firebase project g1-po-app-77790) for local backend development.
    - Frontend: .env.development and .env.production in order-processing-app-frontend for VITE_API_BASE_URL.
- **Firebase Project Setup:**
    - Web app registered in the Firebase project g1-po-app-77790.
    - Firebase Authentication enabled, with Google Sign-In as a provider.
- **Service Account Key (Firebase Admin for setAdminClaim.cjs):**
    - Generated from Firebase Console for project g1-po-app-77790. **EXTREMELY SENSITIVE. MUST NOT BE COMMITTED TO GIT.** Store securely. Used by setAdminClaim.cjs.

7. Authentication & Authorization Flow (NEW - Verified Working)

1. **User Navigates to App:**
    - If the user visits a protected route (e.g., /dashboard) and is not authenticated, ProtectedRoute.jsx redirects them to /login.
    - If they are already authenticated (Firebase session persists), AuthContext sets currentUser, and ProtectedRoute.jsx allows access.
2. **Login (Login.jsx):**
    - User clicks the "Sign In with Google" button.
    - signInWithGoogle() from AuthContext is called.
    - Firebase handles the Google Sign-In flow.
3. **Token Issuance:**

    - Upon successful Google Sign-In, Firebase Authentication mints an ID Token (JWT) for the user (audience: g1-po-app-77790).
    - onAuthStateChanged in AuthContext.jsx fires, setting currentUser.
4. **API Requests from Frontend:**
    - React component gets currentUser from useAuth().
    - Calls await currentUser.getIdToken(true) to get a fresh ID Token.
    - ID Token included in Authorization: Bearer <ID_TOKEN> header.
5. **Backend Token Verification (app.py - @verify_firebase_token decorator):**
    - Flask backend receives the request.
    - Decorator extracts the 'Bearer' token.
    - Uses firebase_auth.verify_id_token(id_token) (initialized for project g1-po-app-77790).
    - Verifies signature, expiration, and audience (aud claim must be g1-po-app-77790).
    - If verification fails, a 401 Unauthorized error is returned.
6. **Backend Authorization (Custom Claim Check):**
    - If the ID token is valid, the decorator inspects the decoded_token for a custom claim: isApproved: true.
    - If missing or not true, a 403 Forbidden error is returned.
    - If present and true, the request proceeds. User info (g.user_uid, g.user_email) is available.
7. **User Approval (Setting Custom Claims):**
    - Manual process using setAdminClaim.cjs script with a service account key for project g1-po-app-77790.

8. Key Features Implemented / Progress Details (Summary)

- Core order processing workflow (PO, Packing Slip, UPS Label generation, GCS upload, supplier email, BigCommerce update): Largely functional.
- Daily IIF Generation for QuickBooks Desktop POs: COMPLETED & VERIFIED.

- Automated triggering of IIF generation via Cloud Scheduler: SETUP & VERIFIED (auth fixed).
- User Authentication via Firebase (Google Sign-In): IMPLEMENTED & VERIFIED (project ID mismatch resolved).
- Backend API protection: All relevant API endpoints require a valid Firebase ID Token (except IIF trigger).
- Basic Authorization: Access restricted to users with an isApproved: true custom Firebase claim.
- Order Ingestion:
    - Preservation of manual statuses (e.g., "pending") fixed by addressing case sensitivity.
    - Stripping of ********** text ********** from customer notes implemented.
- Frontend UI:
    - Global button styling implemented for consistent look and feel.
    - Order Detail page:
        - Spare part SKU lookup and display implemented.
        - Cosmetic fixes for SKU list (no bullets, no "Spare:" prefix, trimmed spaces in links).
        - "BACK TO DASHBOARD" button added.
        - "Process Manually" link centered.
    - Page title and favicon set.
- Resolution of critical backend bugs (UPS state code, ingest_orders status preservation, IIF trigger auth, Firebase Admin SDK init).

9. Critical Next Steps / Remaining MVP Tasks (Updated Plan)

This section outlines tasks to complete the MVP and further enhance the application. Items marked (DONE - Current Chat) were addressed in the latest development session.

9.1. Finalize Authentication & Authorization

1. **Thorough Testing:** (Largely done, but a final round is good)

- **Action:** Test all login, logout, and session persistence scenarios.
- **Action:** Verify that all protected routes correctly redirect unauthenticated users.
- **Action:** Test access with an **approved** Google account - ensure all functionalities work.
- **Action:** Test access with a **non-approved** Google account - ensure backend returns 403 and frontend handles this gracefully.
- **Action:** Test how the application behaves if an ID token expires mid-session.

2. **Robust Frontend Error Handling for 401/403:**
    - **Action:** In all API call sites, enhance error handling for 401/403. Display clear messages, consider auto-logout for 401s.
3. **Admin User Approval Process:**
    - **Current:** Manual setAdminClaim.cjs script.
    - **Action (MVP):** Provide clear, step-by-step documentation for the administrator on how to use setAdminClaim.cjs securely.
    - **Action (Future Enhancement):** Design and implement a simple, secure admin interface within the G1 PO App itself (e.g., /admin/users) for an admin user (identified by isAdmin: true claim) to manage isApproved claims.
4. **Review Cloud Run Service Account IAM Permissions:** (Largely done for Firebase)
    - **Action:** Re-verify that the Cloud Run service account (992027428168-compute@developer.gserviceaccount.com) has only the *necessary* permissions on Firebase project g1-po-app-77790 (e.g., "Firebase Authentication Admin" or similar for token verification) and other GCP services (Cloud SQL, GCS, Secret Manager). Principle of least privilege.

9.2. Complete Frontend Integration & Refinements

1. **Audit All API-Calling Components:**
    - **Action:** Ensure every React component making API calls uses useAuth(), sends the ID token, handles loading states (authLoading, component-specific loading), disables interactive elements if !currentUser, and gracefully handles API errors (especially 401/403). This includes SupplierList.jsx, SupplierForm.jsx, ProductMappingList.jsx, etc.

2. **Spare Part Lookup - Backend Data for hpe_pn_type:**
    - **Action (Backend):** Ensure the /api/orders/<id> endpoint (in get_order_details function in app.py) reliably returns the hpe_pn_type for the *original_sku* of each line item. This was partially addressed by adding a join, but verify its completeness and accuracy for all cases. This is crucial for the frontend to correctly identify which SKUs are "option" types needing a spare part lookup.
3. **Loading Indicators for Spare Part Lookup:**
    - **Action (Frontend):** Implement a visual loading indicator (e.g., a small spinner next to each line item or a general message) in OrderDetail.jsx while spare parts are being fetched via the loadingSpares state.

9.3. PO Export Feature (Excel - Backend & Frontend)

- **Status:** PENDING.
- **Goal:**
    - **Backend:** Create GET /api/exports/pos endpoint to generate and return an Excel (.xlsx) file summarizing Purchase Order details.
    - **Frontend:** Add a button/link on Dashboard.jsx (perhaps with date range filters) to trigger this export.
- **Technical Details:**
    - Backend: Use Flask's send_file or make_response. Library: openpyxl.
    - SQL Query: Join purchase_orders, po_line_items, suppliers.
    - Frontend: API call, handle file download response (Blob, URL.createObjectURL).

9.4. Comprehensive End-to-End Testing

- **Status:** PARTIALLY DONE. FORMAL & FULL TESTING PENDING.
- **Action Items:**
    - Test full order processing flows with authenticated (and authorized) users.
    - Test all edge cases for API integrations (UPS, BigCommerce, Postmark).
    - Thoroughly test all frontend interactions, forms, and error displays across different user auth states.

- User Acceptance Testing (UAT) with the end-user(s).

9.5. Data Integrity for QuickBooks Lists (TERMS and SHIPVIA - Future Enhancement)

- **Status:** Currently N/A (blanked in IIF).
- **Goal (If desired later):** Populate TERMS and SHIPVIA in IIF.
- **Action:** Requires ensuring data matches QuickBooks lists exactly. May involve UI changes for data entry or backend mapping.

9.6. Postmark - External Domain Sending Approval

- **Status:** PENDING VERIFICATION.
- **Action Item:** Ensure Postmark account is fully approved for sending to external supplier domains (DNS changes: DKIM, SPF, Custom Return-Path for sending domain).

9.7. Frontend - Eloquia Font Issue

- **Status:** PENDING.
- **Action Items:** Investigate and resolve font rendering issue on deployed Firebase site. Ensure font files are correctly referenced and served.

9.8. Cloudflare DNS for g1po.com & api.g1po.com

- **Status:** PENDING.
- **Action Items:**
  - Frontend: Add g1po.com as custom domain in Firebase Hosting, update DNS in Cloudflare.
  - Backend (Optional but Recommended): Set up api.g1po.com pointing to Cloud Run service, map custom domain in Cloud Run, update VITE_API_BASE_URL in frontend.

9.9. Further Security Hardening & Best Practices

- **Cloud Run Ingress:**
  - Currently uses --allow-unauthenticated at the Cloud Run service level for the main backend service, with authentication handled at the application level by @verify_firebase_token. The IIF trigger endpoint is also unauthenticated at the Cloud Run level but relies on IAM for Cloud Scheduler invocation.

    - **Action:** Evaluate if this is sufficient for production or if GCP-level authentication (e.g., requiring an OIDC token for all calls, potentially via IAP - Identity-Aware Proxy) should be layered on top for the main service. Application-level auth is a strong start.
- **Input Validation:**
    - **Action:** Re-emphasize and conduct a thorough review of all backend API endpoints for robust input validation on all incoming data (query parameters, JSON bodies).
- **Error Handling & Logging:**
    - **Action:** Enhance user-facing error messages on the frontend for clarity.
    - **Action:** Implement more structured and detailed logging on the backend (Python's logging module, levels, effective use of Google Cloud Logging).

9.10. Supplier and Product (HPE Mapping & QB Mapping) Management UI

- **Status:** PENDING (Basic forms might exist from previous work).
- **Goal:** Develop user-friendly frontend interfaces for CRUD operations on suppliers, hpe_part_mappings, hpe_description_mappings, qb_product_mapping tables.
- **Action Items:** Design and implement React components, connect to backend CRUD APIs (ensuring these APIs are also protected by @verify_firebase_token).

9.11. Unit/Integration Tests (Backend & Frontend)

- **Status:** PENDING.
- **Action Item:** Add automated tests (e.g., pytest for backend, Jest/React Testing Library for frontend) for critical logic.

10. Known Issues / Considerations (Updated)

- **IIF Import Sensitivities:** QuickBooks Desktop IIF import is very sensitive.
- **UPS Production Costs:** Real costs will be incurred with production UPS API.
- **Postmark Deliverability:** DNS setup is crucial.
- **Scalability of Synchronous Processing:** Current order processing is synchronous. Consider asynchronous task queues (e.g., Cloud Tasks) for high volumes or long-running operations within the /process endpoint.

- **Environment Management:**
    - Ensure robust separation for local dev, staging (if any), and production.
    - Frontend VITE_API_BASE_URL needs to be correct for each environment.
    - Backend .env / Secret Manager for credentials.
- **Firebase Project Configuration:** frontend/src/firebase.js is client-side. Ensure the correct Firebase project (g1-po-app-77790) config is used.
- **Custom Claim Management:** The current setAdminClaim.cjs script is manual. A more user-friendly admin interface is a future enhancement.
- **Firebase Quotas:** Monitor Firebase Authentication quotas in the Firebase and Google Cloud Consoles. The auth/quota-exceeded error was resolved, but efficient API call patterns are important.
- **Service Account Key Security:** The Firebase Admin SDK service account key (for setAdminClaim.cjs) is highly sensitive. **Never commit to Git.** For local backend development, use GOOGLE_APPLICATION_CREDENTIALS. Cloud Run uses its runtime service account.
- **Spare Part Lookup Data Dependency:** The accuracy of the spare part lookup feature heavily depends on the hpe_part_mappings table being accurate and the /api/orders/<id> endpoint correctly providing the hpe_pn_type for each original SKU.

11. Environment Setup Guide (Recap & Update for New Developer)

11.1. Backend (Python/Flask - order-processing-app directory)

1. Clone repository.
2. Navigate to backend root (order-processing-app).
3. Ensure Python 3.9+ is installed.
4. Create/activate Python virtual environment (e.g., python -m venv .venv then source .venv/bin/activate or .\.venv\Scripts\activate).
5. Install dependencies: pip install -r requirements.txt.
6. Create .env file in backend root (from template, **DO NOT COMMIT SECRETS**). Populate with:
    - Database credentials (for local dev, often connect directly or via local Cloud SQL Proxy).

- BigCommerce API credentials.
- UPS API credentials.
- Postmark Server API Token, Sender/BCC addresses.
- GCS Bucket Name, Company Logo GCS URI.
- GOOGLE_APPLICATION_CREDENTIALS: Path to your downloaded Firebase Admin SDK service account key JSON file (from Firebase project g1-po-app-77790). This allows the local Flask app to initialize firebase-admin.
- Ship From address details.
- Other config like BC_PROCESSING_STATUS_ID, QUICKBOOKS_EMAIL_RECIPIENT, etc.

7. For Local DB (Cloud SQL Proxy):
    - Install gcloud CLI and authenticate (gcloud auth login, gcloud auth application-default login).
    - Download/run Cloud SQL Auth Proxy, pointing to the production Cloud SQL instance.
    - Configure .env DB variables for proxy connection (host 127.0.0.1, port 5432).
8. Run Flask app locally: flask run or python app.py.

11.2. Frontend (React/Vite - order-processing-app-frontend directory)

1. Clone repository.
2. Navigate to frontend root (order-processing-app-frontend).
3. Ensure Node.js (LTS version) and npm (or yarn) are installed.
4. Install dependencies: npm install (or yarn install).
5. **Firebase Setup (src/firebase.js or src/config/firebaseConfig.js):**
    - Ensure this file contains the Firebase project configuration for project g1-po-app-77790 (apiKey, authDomain, projectId, etc.).
    - Initializes Firebase app and exports auth service.
6. Create .env.development in frontend root for local API URL:
    - VITE_API_BASE_URL=http://localhost:8080/api (or your local backend URL).

7. Create .env.production for deployed API URL:
    - VITE_API_BASE_URL=https://g1-po-app-backend-service-992027428168.us-central1.run.app/api
8. Run Vite dev server: npm run dev.

12. Deployment Process (Recap & Update)

12.1. Backend (to Cloud Run)

1. Make code changes in order-processing-app.
2. Update requirements.txt if new Python packages are added.
3. Rebuild Docker image: docker build -t gcr.io/YOUR_GCP_PROJECT_ID/g1-po-app-backend-service:YOUR_TAG . (where YOUR_GCP_PROJECT_ID is for order-processing-app-458900).
4. Push Docker image: docker push gcr.io/YOUR_GCP_PROJECT_ID/g1-po-app-backend-service:YOUR_TAG
5. Redeploy Cloud Run service:
6. gcloud run deploy g1-po-app-backend-service \
7. --image gcr.io/YOUR_GCP_PROJECT_ID/g1-po-app-backend-service:YOUR_TAG \
8. --region YOUR_REGION \
9. --allow-unauthenticated \ # For user-facing APIs; IIF trigger is IAM secured
10. --set-env-vars "GOOGLE_APPLICATION_CREDENTIALS=" \ # Rely on service account identity
11. # ... include all other necessary flags for env vars from Secret Manager, Cloud SQL instance, etc.

    - The runtime service account for Cloud Run (992027428168-compute@developer.gserviceaccount.com) must have permissions on Firebase project g1-po-app-77790 for token verification.

12.2. Frontend (to Firebase Hosting)

1. Make code changes in order-processing-app-frontend.

2. Ensure VITE_API_BASE_URL in .env.production points to the correct deployed backend URL.
3. Build for production: npm run build.
4. Deploy to Firebase: firebase deploy --only hosting (or firebase deploy --only hosting:g1-po-app-77790 to be explicit).

13. Conclusion for Handover

Significant progress has been made on the G1 PO App. It now boasts a functional core backend, automated daily IIF generation, and a robust, verified user authentication (Google Sign-In via Firebase project g1-po-app-77790) and authorization (custom claims) system. The frontend has been enhanced with features like spare part lookups, improved data display, and consistent global UI styling for buttons. Critical bugs related to authentication, data integrity during ingestion, and scheduled task execution have been resolved.

The immediate next steps involve thorough end-to-end testing of all integrated features, completing the PO export functionality, and then moving towards further UI enhancements for supplier and product management. This document, along with the codebase and existing cloud infrastructure, should provide the incoming developer with a solid and detailed foundation to successfully complete the MVP and move the application towards full production readiness.