G1 PO App - Consolidated Handover Report & Project Plan (v16.0)

Date: May 20, 2025 (Reflecting all progress and chat sessions ending May 20, 2025, 1:19 PM CDT) **Version:** 16.0 (This document supersedes v14.0 and incorporates all subsequent development, refactoring, troubleshooting, and planning from our recent chat sessions.) **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 (including significant backend refactoring), architecture, recent progress, known issues, and a detailed plan for future development and completion. It is intended to facilitate a smooth handover to a human developer and support continued development of the application.

Table of Contents

---

1. Project Goal & Core Value Proposition

(As per v14.0) 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 (Summarized from v14.0):

- Ingest orders from BigCommerce.
- Manage orders via a web interface.
- Generate documentation (POs, Packing Slips, UPS & FedEx Shipping Labels).
- Facilitate supplier communication.
- Update BigCommerce (shipment details, order statuses).
- Integrate with QuickBooks Desktop (POs, Sales Orders/Invoices & Payments via IIF files).

---

2. Current State of the Application (As of May 20, 2025)

The application has many core features implemented and has recently undergone a significant backend refactoring. After a series of troubleshooting steps, the refactored backend has been successfully deployed to Cloud Run.

- **Key Achievements & Functionalities (Consolidated & Updated from v14.0):**
    - Most functionalities detailed in Handover Report 20250519b.pdf (v14.0, Section 2, pages 3-7) remain relevant. This includes robust order ingestion (including parsing for third-party shipping accounts and BigCommerce billing address/shipping cost storage), various fulfillment modes (Single/Multi-Supplier POs, G1 Onsite Fulfillment, Customer-Billed Shipments), PDF document generation (POs, Packing Slips), functional UPS label generation ("Bill Sender" and "Bill Third Party"), groundwork for FedEx label generation (OAuth working, "Bill SENDER" tested in production with test-marked label, "Bill RECIPIENT" sandbox blocked), email communications via Postmark, BigCommerce integration for order data and updates, and extensive QuickBooks Desktop IIF file generation capabilities for Purchase Orders,

Sales Orders (as Invoices), and corresponding Payments, including US Central Time conversion and specific formatting.

- Firebase Authentication (Google Sign-In with isApproved custom claim) is implemented.
- The frontend features a Dashboard, a detailed OrderDetail page, and a Utilities section (linking to Standalone UPS Labeler, QuickBooks Sync, Supplier Management, HPE Description Management). Dark mode CSS is implemented for forms.

- **Recent Backend Refactoring: Introduction of Blueprints (This Session):**
  - The main backend Flask application file, app.py, has been refactored to improve organization, maintainability, and scalability.
  - API route logic has been moved from app.py into separate Python modules (Blueprints) located within a new order-processing-app/blueprints/ directory.
  - The main app.py file now focuses on:
    - Flask app initialization and core configurations (CORS, debug mode).
    - Loading environment variables and secrets.
    - Initializing shared services: database engine (engine), Google Cloud Storage client (storage_client), Firebase Admin SDK.
    - Defining globally used helper functions (e.g., convert_row_to_dict, make_json_safe) and the verify_firebase_token decorator.
    - Importing and registering all blueprint objects.
    - Retaining only basic routes like / and /test_db.
  - Service modules (document_generator.py, shipping_service.py, email_service.py, iif_generator.py) remain in the root of order-processing-app and are imported directly by app.py or the blueprint modules as needed.
  - Import statements within blueprint files were corrected from from ..app import ... to from app import ... and from .. import service_module to import service_module to resolve ImportError issues in the Cloud Run environment.
- **Deployment Status & Recent Troubleshooting Summary (This Session):**

- The application backend, with the new blueprint structure, is **currently deployed and running successfully on Cloud Run**.
- Achieving this successful deployment involved several troubleshooting steps:
    - Resolving local Windows "Fatal error in launcher" for pip and gunicorn by recreating the Python virtual environment (venv) within the project's current path.
    - Setting PowerShell execution policy (Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser) to allow venv activation.
    - Correcting Python relative import errors in blueprint files for the Cloud Run environment.
    - Diagnosing "container failed to start" errors on Cloud Run, where initially no application logs were visible. This involved:
        - Ensuring entrypoint.sh used LF (Unix/Linux) line endings.
        - Testing with a progressively more verbose and simplified entrypoint.sh to confirm script executability and basic logging.
        - The current entrypoint.sh uses a Gunicorn command with verbose logging flags (--log-level debug, --access-logfile '-', --error-logfile '-') and simplified worker settings (--workers 1 --threads 2), which led to a successful deployment.
- A frontend error Error: Firebase: Error (auth/quota-exceeded). was observed, which is separate from the backend deployment and relates to Firebase service quotas.

---

3. Technology Stack & Project Layout (Updated)

(Based on v14.0 and current session changes)

- **Backend (order-processing-app directory):**
    - Language/Framework: Python 3.9+, Flask
    - WSGI Server: Gunicorn

- Key Libraries: SQLAlchemy, psycopg2-binary (via google-cloud-sql-connector), requests, python-dotenv, ReportLab, Pillow, google-cloud-sql-connector, google-cloud-storage, firebase-admin, postmarker, pytz.
- **Updated Directory Structure:**
- order-processing-app/
- ├── .env
- ├── requirements.txt
- ├── Dockerfile
- ├── entrypoint.sh
- ├── app.py # Main Flask app, shared resources, blueprint registration
- ├── blueprints/
- │ ├── __init__.py
- │ ├── hpe_mappings.py
- │ ├── orders.py
- │ ├── quickbooks.py
- │ ├── reports.py
- │ ├── suppliers.py
- │ └── utils_routes.py
- ├── document_generator.py
- ├── email_service.py
- ├── iif_generator.py
- └── shipping_service.py

- **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 (e.g., OrderDetail.css).
- **Database:** PostgreSQL on Google Cloud SQL.
- **Cloud Platform (GCP):** Cloud Run, Cloud SQL, Artifact Registry, Secret Manager, Cloud Storage, Cloud Scheduler, Cloud Logging, IAM.
- **Authentication:** Firebase Authentication (Google Sign-In, Custom Claims: isApproved: true).
- **External APIs:** BigCommerce API, UPS API (OAuth 2.0), FedEx API (OAuth 2.0), Postmark API.

---

4. Key Code Files & Their Roles (Updated for Refactoring)

- **Backend:**
    - **app.py**: Core application setup. Initializes Flask, loads all configurations, sets up database connections, GCS client, Firebase Admin. Defines shared helper functions and the verify_firebase_token decorator. Imports and registers all blueprints from the blueprints/ directory. Contains only minimal root routes (/, /test_db).
    - **blueprints/__init__.py**: Empty file, makes the blueprints directory a Python package.
    - **blueprints/hpe_mappings.py**: Defines Flask Blueprint for HPE PO description mappings CRUD operations (/api/hpe-descriptions/*) and related SKU lookups (/api/lookup/*).
    - **blueprints/orders.py**: Defines Flask Blueprint for all order-related operations, including fetching orders, order details, status updates, BigCommerce order ingestion (/api/ingest_orders), and processing orders (/api/orders/:order_id/process).
    - **blueprints/quickbooks.py**: Defines Flask Blueprint for QuickBooks integration tasks, including scheduled and user-triggered IIF batch generation (/api/tasks/*) and the on-demand full sync endpoint (/api/quickbooks/trigger-sync).
    - **blueprints/reports.py**: Defines Flask Blueprint for generating reports, currently the daily revenue report (/api/reports/daily-revenue).

- **blueprints/suppliers.py**: Defines Flask Blueprint for supplier CRUD operations (/api/suppliers/*).
- **blueprints/utils_routes.py**: Defines Flask Blueprint for miscellaneous utility endpoints, like the standalone UPS label generator (/api/utils/generate_standalone_ups_label).
- **Service Modules** (document_generator.py, email_service.py, iif_generator.py, shipping_service.py): These modules contain the business logic for their respective domains and are imported by app.py or the blueprint files as needed. Their roles are unchanged from v14.0.
- **Dockerfile**:

Dockerfile

```
# Step 1: Specify the Base Image

FROM python:3.9-slim


# Step 2: Set the Working Directory

WORKDIR /app


# Step 3: Copy Dependencies File

COPY requirements.txt .


# Step 4: Install Dependencies

RUN pip install --no-cache-dir -r requirements.txt


# Step 5: Copy Application Code

COPY . .


# Step 6: Expose the Port (Cloud Run will use the PORT env var)

EXPOSE 8080
```

```
# Step 7: Define Environment Variable for the Port (Cloud Run will override this with its own)
ENV PORT=8080

# Step 8: Copy entrypoint script and make it executable
COPY entrypoint.sh /entrypoint.sh
RUN chmod +x /entrypoint.sh

# Step 9: Set the Entrypoint
ENTRYPOINT ["/entrypoint.sh"]
```

- **entrypoint.sh** (current version that led to successful deployment):

```
Bash
#!/bin/sh
set -ex # Add -x for xtrace to see commands being executed

echo "ENTRYPOINT: Script starting."
echo "ENTRYPOINT: Current directory: $(pwd)"
echo "ENTRYPOINT: Listing /app directory contents:"
ls -la /app
echo "ENTRYPOINT: PORT environment variable is: '$PORT'"
echo "ENTRYPOINT: PYTHONPATH is: '$PYTHONPATH'"
echo "ENTRYPOINT: PATH is: '$PATH'"
echo "ENTRYPOINT: Python version: $(python --version || echo 'Failed to get Python version')"
echo "ENTRYPOINT: Attempting to get Gunicorn version..."
gunicorn --version || echo "ENTRYPOINT: gunicorn command failed or not found"
```

```
echo "ENTRYPOINT: Attempting to start Gunicorn with detailed logging..."

exec gunicorn \
  --workers 1 \
  --threads 2 \
  --bind "0.0.0.0:$PORT" \
  --log-level debug \
  --access-logfile '-' \
  --error-logfile '-' \
  "app:app"

echo "ENTRYPOINT: Gunicorn exec command finished OR FAILED TO EXEC."
exit 1
```

- **Frontend:**
    - OrderDetail.jsx and OrderDetail.css: Core component for displaying and processing individual orders. Recent discussions focused on UI improvements for its success message and a "See Quotes" feature.
    - Other key components (App.jsx, Dashboard.jsx, AuthContext.jsx, utility forms like HpeDescriptionForm.jsx, EditHpeDescriptionForm.jsx, HpeDescriptionList.jsx, UtilitiesLandingPage.jsx) remain as described in v14.0 unless updated through direct file uploads in this session.

---

5. Database Schema Notes & Key Tables

(No changes to the database schema were made in this session. Refer to Section 5 of Handover Report 20250519b.pdf (pages 13-15) for detailed schema notes on tables: orders, order_line_items, suppliers, purchase_orders, po_line_items, shipments, hpe_part_mappings, hpe_description_mappings, and qb_product_mapping. The QuickBooks sync status columns for orders and purchase_orders tables are defined there and are key for the IIF generation logic.)

---

6. Developer Setup & Environment (Updated with Troubleshooting Notes)

(Refer to Section 6 of Handover Report 20250519b.pdf (pages 16-18) for the base setup. Key updates and reminders from this session:)

- **Backend (order-processing-app):**
    - **Virtual Environment (venv):** Essential for dependency management.
        - If "Fatal error in launcher" occurs for pip or gunicorn on Windows, it often indicates corrupted venv paths. **Solution:** Deactivate, delete the venv folder, recreate it (python -m venv venv), activate, and then reinstall dependencies (python -m pip install -r requirements.txt).
    - **PowerShell Execution Policy (Windows):** If .\venv\Scripts\activate.ps1 is blocked, run: Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser in PowerShell (as admin if setting for LocalMachine).
    - **Gunicorn on Windows (Local Testing):** Gunicorn may error with ModuleNotFoundError: No module named 'fcntl' as fcntl is Unix-specific. This will **not** affect Cloud Run (Linux). For local WSGI testing on Windows, use Flask's built-in dev server (flask run) or a Windows-compatible server like Waitress (pip install waitress; waitress-serve --port=8080 app:app).
    - **Line Endings for Shell Scripts:** Ensure .sh files like entrypoint.sh are saved with LF (Unix/Linux) line endings, not CRLF (Windows), to prevent execution issues in the Linux container. VS Code allows easy viewing and changing of line endings in the status bar.
- **Frontend (order-processing-app-frontend):** Standard Node.js/npm setup, build with npm run build.
- **Deployment Commands:**
    - **Docker Authentication (one-time for a region):** gcloud auth configure-docker YOUR_REGION-docker.pkg.dev
    - **Backend (Cloud Run):**

1. docker build -t YOUR_IMAGE_NAME . (e.g., gcr.io/YOUR_PROJECT_ID/YOUR_SERVICE_NAME:TAG)

2. docker tag YOUR_IMAGE_NAME YOUR_REGION-docker.pkg.dev/YOUR_PROJECT_ID/YOUR_REPO_NAME/YOUR_IMAGE_NAME:TAG

3. docker push YOUR_REGION-docker.pkg.dev/YOUR_PROJECT_ID/YOUR_REPO_NAME/YOUR_IMAGE_NAME:TAG

4. Use the comprehensive gcloud run deploy ... command provided by the user, ensuring all environment variables and secrets are correctly specified. (The last one used was successful for service g1-po-app-backend-service with image tag 20250520b which led to revision ...00212-hwh after entrypoint.sh fixes).

    - **Frontend (Firebase Hosting):** firebase deploy --only hosting

- **Crucial Environment Variables:** Refer to Handover Report 20250519b.pdf (Section 6, pages 16-17) and the user's gcloud run deploy command for the extensive list of required backend environment variables (set via --set-env-vars or --update-secrets). For frontend, VITE_API_BASE_URL and Firebase SDK config are key.

---

7. Summary of Recent Changes (From Current Chat Session - May 20, 2025)

- **Backend Refactoring:** Successfully restructured the main app.py by moving API route logic into separate Flask Blueprint files within a blueprints/ directory. Updated all import statements in blueprints to correctly reference shared objects from the main app.py and service modules.
- **Deployment Troubleshooting:**
    - Resolved local Windows venv pathing issues ("Fatal error in launcher") by recreating the virtual environment.
    - Addressed PowerShell execution policy issues preventing venv activation.
    - Fixed ImportError: attempted relative import beyond top-level package in Cloud Run by correcting Python import paths in blueprint files.
    - Systematically debugged "container failed to start" errors on Cloud Run by:
        - Verifying and ensuring LF (Linux) line endings for entrypoint.sh.
        - Using a simplified entrypoint.sh to test basic script execution and logging in Cloud Run, which successfully produced output.

        - Modifying the main entrypoint.sh to include verbose Gunicorn logging flags (--log-level debug, outputting access and error logs to stdout/stderr) and simplified worker settings.
- **Successful Cloud Run Deployment:** The backend application with the refactored structure was successfully deployed and started on Cloud Run using the verbose Gunicorn entrypoint.
- **Frontend (OrderDetail.jsx):**
    - Discussed UI enhancements for the "Order Processed Successfully" screen to add a link to the BigCommerce order dashboard and modified the "See Quotes" button functionality to only copy a search phrase to the clipboard.
    - Provided updated code for OrderDetail.jsx to implement these changes.
- **Deferred Feature:** The plan to add CRUD functionality for the qb_product_mapping table was discussed but set aside for future implementation.

---

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

- **Phase 0: Immediate Post-Handover Validation & Critical Fixes**
    1. **Thorough API Testing (Critical):**
        - **Action:** Systematically test all API endpoints (GET, POST, PUT, DELETE for all resources like orders, suppliers, HPE mappings, QuickBooks triggers, etc.) using the deployed frontend or a tool like Postman/Insomnia.
        - **Goal:** Verify that the backend refactoring into blueprints did not introduce any regressions in API behavior or data handling. Confirm all functionalities are working as before.
    2. **Monitor Cloud Run Logs of Successful Deployment:**
        - **Action:** While testing, keep a close eye on the Cloud Run logs for the current successfully deployed revision (e.g., ...00212-hwh or the latest after further minor tweaks to entrypoint.sh if any).
        - **Goal:** Ensure the application is stable under use and identify any runtime errors or unexpected behavior not caught during startup.

3. **Investigate & Resolve Firebase auth/quota-exceeded Error (Critical for UI Testing):**
    - **Action:** Access the Google Cloud Console for project order-processing-app-458900. Navigate to "IAM & Admin" > "Quotas" and search for "Identity Platform" and "Firebase Authentication" related quotas. Analyze usage against limits.
    - **Goal:** Determine the specific quota being exceeded and take corrective action (e.g., if DAU for Identity Platform, or token service operations). This will likely require upgrading the Firebase project to the "Blaze" (pay-as-you-go) plan to ensure uninterrupted authentication services.
4. **Review & Tune Gunicorn Production Settings in entrypoint.sh:**
    - **Action:** The current entrypoint.sh uses --workers 1 --threads 2 --log-level debug. Once extensive testing confirms stability, adjust these for production. Consider increasing threads (e.g., --threads 4 or --threads 8) or workers if appropriate for the instance CPU. Change --log-level to info or warning for production to reduce log volume.
    - **Goal:** Optimized and stable Gunicorn configuration for the production Cloud Run environment.

- **Phase 1: Frontend UI Enhancements (OrderDetail.jsx)**
    1. **Mobile View - Status Alignment:** (As discussed, user to verify if existing CSS is sufficient or needs minor tweaks)
        - **Action:** Confirm if the order status badge on the OrderDetail.jsx page aligns to the right on the same line as the order title on mobile. If not, adjust CSS for .order-title-section (likely with Flexbox) as previously discussed.
        - **Goal:** Improved mobile layout for order status display.
    2. **Desktop View - Fix Duplicate Status:** (Needs user clarification on where the duplicate appears)
        - **Action:** Based on user feedback identifying where the duplicate order status appears on desktop in OrderDetail.jsx, investigate and remove the redundant rendering.

        - **Goal:** Clean and non-redundant status display.
    3. **Desktop View - "Table" to Cards Layout:** (Needs user clarification)
        - **Action:** Discuss with the user which specific "table" elements on OrderDetail.jsx they want changed to cards. If it's the overall arrangement of sections (<section class="card">), implement a multi-column responsive layout using CSS Grid or Flexbox. If it's for item lists within forms, evaluate design and implement accordingly.
        - **Goal:** More modern and consistent card-based UI on desktop.
    4. **Verify "Copy Gmail Search Phrase" Button:**
        - **Action:** Test the "Copy Gmail Search Phrase" button on the "RFQ Sent" order view in OrderDetail.jsx to ensure it correctly copies the phrase to the clipboard and provides appropriate user feedback.
        - **Goal:** Functional "See Quotes" helper.
    5. **Implement Order Processed Success Screen Update:**
        - **Action:** Verify the implemented changes in OrderDetail.jsx for the order processing success message, ensuring the "G1 BC Order Dashboard" link is present and functional.
        - **Goal:** User-friendly success screen with a clear link to the BigCommerce dashboard.

- **Phase 2: FedEx Integration Finalization** (Carried over from v14.0)
    - Task 1: FedEx "Bill RECIPIENT" Sandbox Testing (Currently Blocked - Requires FedEx Support).
    - Task 2: FedEx Production Label Certification (Critical, Pending).
    - Task 3: Integrate successful FedEx Label Generation (including "Bill Recipient") into blueprints/orders.py (process_order_route) post-certification.
    - Task 4: (Optional) Standalone FedEx Label Generator UI & Backend (blueprints/utils_routes.py and new frontend component).
    - Task 5: Production "Bill Recipient" Testing with real customer accounts (Pilot).

- **Phase 3: QuickBooks Integration - Full Testing & Refinement** (Carried over from v14.0)
    - Task 1: Finalize QuickBooks Version & Integration Strategy Detail (Confirm Desktop, IIF sufficiency vs. QBWC).
    - Task 2: Database Schema for QuickBooks Sync Status (Verify columns exist as per v14.0: qb_sales_order_sync_status, etc. in orders; qb_po_sync_status, etc. in purchase_orders).
    - Task 3 & 4: Backend Logic for Sales Order & Purchase Order Data Extraction (Verify logic in blueprints/quickbooks.py and iif_generator.py correctly fetches pending records).
    - Task 5 & 6: IIF File Generation for Sales Orders & Purchase Orders (Thoroughly test generated IIF files by importing into a TEST QuickBooks Desktop company. Verify all fields, mappings, customer/item creation, tax, shipping, payment application).
    - Task 7: Backend API Endpoint for On-Demand QuickBooks Sync (/api/quickbooks/trigger-sync in blueprints/quickbooks.py): Ensure it correctly calls IIF generation, emails files, and accurately updates qb_..._sync_status and qb_..._synced_at in the database using returned DB IDs. Implement robust error logging to qb_..._last_error.
    - Task 8: Frontend UI for QuickBooks Sync (QuickbooksSync.jsx): Enhance to display actual last sync timestamp, counts of pending POs/Sales Orders (requires new backend endpoints in blueprints/quickbooks.py), and provide detailed user feedback/error messages.
    - Task 9: Comprehensive end-to-end testing of the QuickBooks sync process with various scenarios.
- **Phase 4: Deferred/Future Utility - QuickBooks Product Mapping CRUD** (From this session, user requested to set aside for now)
    - **Action:** Implement backend API endpoints (e.g., in a new blueprints/qb_product_mappings.py) for CRUD operations on the qb_product_mapping table (option_pn to qb_item_name).
    - **Action:** Develop frontend components (QbProductMappingList.jsx, QbProductMappingForm.jsx, EditQbProductMappingForm.jsx) similar to the HPE description management.

    - **Action:** Add new routes in App.jsx for these components.
    - **Action:** Add a new card/link to UtilitiesLandingPage.jsx.
    - **Goal:** Allow admin users to manage QuickBooks item mappings directly in the application.
- **Phase 5: MVP Task Completion** (Consolidated from v14.0)
    - PO Export Feature (Excel - Backend in blueprints/reports.py or a new export blueprint & Frontend).
    - Finalize Authentication & Authorization Robustness (Consistent frontend 401/403 handling from apiService, review backend @verify_firebase_token usage).
- **Phase 6: Infrastructure & Deployment Finalization** (Consolidated from v14.0)
    - Postmark - External Domain Sending Approval (DKIM, SPF, Custom Return-Path).
    - Cloudflare DNS for Production Domains (If using custom domains for Firebase Hosting/Cloud Run).
- **Phase 7: Long-Term Improvements & Best Practices** (Consolidated from v14.0)
    - Backend Daily Revenue Aggregation Timezone: If specific local timezone reporting is required.
    - Security Hardening: Cloud Run Ingress (consider IAP), comprehensive server-side input validation, enhanced structured logging.
    - Asynchronous Task Processing: For process_order_route and batch IIF generation using Google Cloud Tasks.
    - Admin User Approval UI: For Firebase Custom Claims management.
    - Data Integrity for QuickBooks Lists (TERMS, SHIPVIA for IIFs).
    - Automated Unit/Integration Tests (Pytest for backend, Jest/React Testing Library for frontend).
    - Order Comment Parsing Reliability: Evaluate robustness or consider BigCommerce Order Metafields.

---

9. Known Issues & Considerations (Updated & Consolidated)

- **Firebase Authentication Quota Exceeded (NEW - High Priority):**
    - Frontend displays Error: Firebase: Error (auth/quota-exceeded).
    - **Action Needed:** Investigate Firebase project's Authentication/Identity Platform usage quotas. Likely requires upgrading to the Firebase "Blaze" plan.
- **FedEx Integration Blockers:** (As per v14.0)
    - "Bill RECIPIENT" Sandbox Testing: Requires FedEx Developer Support for valid test accounts/procedures.
    - Production Label Certification: Process is pending with FedEx.
- **QuickBooks Desktop & IIF Limitations:** (As per v14.0)
    - IIF is fragile, offers poor error feedback, and cannot easily update existing QB transactions. QBWC or QB Online API are more robust (significant) future alternatives if IIF proves insufficient.
- **Gunicorn fcntl Error on Windows (Local Development):**
    - ModuleNotFoundError: No module named 'fcntl' occurs when running Gunicorn directly on Windows. This is OS-specific and does **not** affect Cloud Run (Linux).
    - **Workaround:** Use Flask's built-in development server (flask run) or waitress for local WSGI testing on Windows.
- **Scalability of Synchronous Processing:** (As per v14.0)
    - Long-running tasks (order processing, IIF generation) are synchronous. Consider asynchronous processing for higher volumes.
- **Environment Management, GCP Quotas, Service Account Key Security (Local Dev), Reporting Timezone (UTC default), BigCommerce API Rate Limits:** These considerations remain relevant as detailed in v14.0.

---

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

The G1 PO App has undergone a significant backend refactoring to improve code organization using Flask Blueprints. After intensive troubleshooting of deployment issues

related to local environments, Python imports, and container startup on Cloud Run, the application with this new structure has been **successfully deployed and is running**. Core functionalities for order processing, document generation, UPS label creation, and initial QuickBooks IIF integration remain in place.

Immediate Priorities for the Incoming Developer are:

1. **Thorough End-to-End Testing:** Critically important to validate all API endpoints and application workflows after the backend refactoring. Monitor Cloud Run logs for the active revision during testing.
2. **Resolve Firebase auth/quota-exceeded Error:** This is essential for unblocking frontend usability and testing. Investigate Firebase project quotas and likely upgrade to the Blaze plan.
3. **Implement Frontend UI Enhancements for OrderDetail.jsx:** Proceed with the discussed changes for mobile status display, fixing any duplicate status on desktop, and transforming the desktop layout to a card-based system as per user requirements. Verify the "Copy Gmail Search Phrase" button. Ensure the "Order Processed Successfully" screen includes the new BigCommerce dashboard link.
4. **Review and Finalize Gunicorn Production Configuration:** The current deployment uses debug logging and minimal workers for Gunicorn. Tune these settings for a production environment once stability is fully confirmed.
5. **Address Other High-Priority Items from the Plan:** Systematically work through the remaining phases, starting with FedEx integration blockers and comprehensive QuickBooks IIF testing and UI finalization.

This consolidated report, along with the refactored codebase (), the Dockerfile and entrypoint.sh scripts, and previous handover documentation, should equip the incoming developer to effectively take ownership of the project and drive it to completion.