G1 PO App - Consolidated Handover Report & Project Plan (v15.0)

Date: May 20, 2025 (Reflecting all progress and chat sessions ending May 20, 2025, CDT)
Version: 15.0 (This document supersedes v14.0 and incorporates all subsequent development, refactoring, troubleshooting, and planning.) **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 (including recent refactoring), 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.

Table of Contents

1. Project Goal & Core Value Proposition
2. Current State of the Application (As of May 20, 2025)
    - Key Achievements & Functionalities (Consolidated & Updated)
    - Recent Backend Refactoring (Blueprints)
    - Deployment Status & Recent Troubleshooting
3. Technology Stack & Project Layout (Updated)
    - Backend (order-processing-app directory)
    - Frontend (order-processing-app-frontend directory)
    - Database
    - Cloud Platform
    - Authentication
    - External APIs
4. Key Code Files & Their Roles (Updated for Refactoring)
    - Backend (Main app.py, Blueprint files, Service modules)
    - Frontend (Key components like OrderDetail.jsx)
5. Database Schema Notes & Key Tables
6. Developer Setup & Environment (Updated with Troubleshooting Notes)

    - Backend
    - Frontend
    - Running Locally (including Gunicorn on Windows considerations)
    - Deployment (Cloud Run & Firebase Hosting)
    - Crucial Environment Variables
7. Summary of Recent Changes (This Chat Session - May 20, 2025)
8. Detailed Plan for Completion & Future Tasks (Updated & Prioritized)
    - Immediate Post-Handover Tasks & Verification
    - Frontend UI Enhancements (OrderDetail.jsx)
    - FedEx Integration Finalization
    - QuickBooks Integration - Full Testing & Refinement
    - MVP Task Completion
    - Infrastructure & Deployment Finalization
    - Long-Term Improvements & Best Practices
9. Known Issues & Considerations (Updated & Consolidated)
10. Conclusion for Handover & Immediate Next Steps for Human Developer

---

1. Project Goal & Core Value Proposition

To develop a secure, internal web application ("G1 PO App") that automates and streamlines the purchase order (PO) and shipment process for drop-shipped and G1-stocked items. The primary aim is to reduce manual effort, improve accuracy, provide a centralized platform for critical business workflows, and enhance operational efficiency, including replacing functionalities of tools like T-HUB for BigCommerce-to-QuickBooks synchronization.

Core Functionality (from v14.0, still applies):

- Ingest orders from BigCommerce.
- Allow authorized users to manage these orders via a web interface.

- Generate necessary documentation (POs, Packing Slips, Shipping Labels for UPS & FedEx).
- Facilitate communication with suppliers (e.g., emailing POs).
- Update BigCommerce with shipment details and order statuses.
- Integrate with QuickBooks Desktop for accounting (Purchase Orders and Sales Orders/Invoices & Payments via IIF files).

---

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

The application is substantially functional with many core features implemented. Significant progress has been made on handling complex order scenarios, robust authentication/authorization, integrations with external services, UI enhancements, administrative utilities, and QuickBooks Desktop integration using IIF files.

- **Key Achievements & Functionalities:**
    - Most functionalities detailed in Handover Report v14.0 (sections 2 and 4) remain relevant. This includes order ingestion, detailed order view, various fulfillment modes (Single/Multi-Supplier POs, G1 Onsite Fulfillment, Customer-Billed Shipments), document generation (POs, Packing Slips), UPS label generation, FedEx integration groundwork, email communications, BigCommerce integration, and initial QuickBooks IIF generation for POs and Sales Orders/Payments.
    - User authentication via Firebase is robust.
    - The frontend UI has a dashboard, order detail page, and a utilities section for admin tasks.
- **Recent Backend Refactoring (Blueprints - This Session):**
    - The main backend Flask application file, app.py, has been significantly refactored for better organization and maintainability.
    - Most API route logic has been moved from app.py into separate Python files (Blueprints) located in a new order-processing-app/blueprints/ directory.
    - The main app.py now primarily handles Flask app initialization, configuration loading (environment variables, secrets), database engine creation, GCS client setup, Firebase Admin SDK initialization, definition of shared helper

functions (e.g., convert_row_to_dict, make_json_safe), the verify_firebase_token decorator, and registration of all blueprints.

- The new blueprint files are:
  - blueprints/__init__.py: Makes the blueprints directory a Python package.
  - blueprints/orders.py: Handles all order-related routes (e.g., get orders, get order details, process order, ingest orders).
  - blueprints/suppliers.py: Handles CRUD operations for suppliers.
  - blueprints/hpe_mappings.py: Handles CRUD for HPE description mappings and related lookup routes.
  - blueprints/quickbooks.py: Handles QuickBooks IIF generation trigger routes (scheduled tasks, on-demand sync).
  - blueprints/reports.py: Handles routes for generating reports (e.g., daily revenue).
  - blueprints/utils_routes.py: Handles utility routes like the standalone UPS label generator.
- Import statements within these blueprint files have been updated to correctly import shared objects from the main app.py (e.g., from app import engine) and service modules (e.g., import shipping_service).

- **Deployment Status & Recent Troubleshooting (This Session):**
  - After the backend refactoring, several deployment issues to Cloud Run were encountered and resolved:
    - Local Windows "Fatal error in launcher" for pip and gunicorn was traced to virtual environment pathing issues, resolved by recreating the venv cleanly within the project directory.
    - PowerShell execution policy preventing venv activation was addressed using Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser.
    - An ImportError: attempted relative import beyond top-level package in Cloud Run logs (due to from ..app import ... in blueprints) was fixed by changing imports to from app import ....

        - Subsequent Cloud Run "container failed to start and listen on the port" errors, where no application logs were visible, were debugged by:
            - Ensuring entrypoint.sh had LF (Linux) line endings.
            - Testing with a simplified entrypoint.sh which confirmed the script *could* run and produce logs.
            - Reinstating the Gunicorn command in entrypoint.sh with highly verbose logging (--log-level debug, --access-logfile '-', --error-logfile '-') and simplified worker settings (--workers 1 --threads 2).
    - The latest deployment with the refactored code and verbose Gunicorn entrypoint.sh was **successful**. The application is currently running on Cloud Run.

---

3. Technology Stack & Project Layout (Updated)

(Largely based on v14.0, with updates for backend structure)

- **Backend (order-processing-app directory):**
    - Language/Framework: Python 3.9+, Flask
    - WSGI Server: Gunicorn (for deployment)
    - Key Libraries: SQLAlchemy, requests, ReportLab, Pillow, google-cloud-sql-connector, google-cloud-storage, firebase-admin, postmarker, pytz.
    - **NEW Structure:**
    - order-processing-app/
    - ├── .env
    - ├── requirements.txt
    - ├── app.py # Main Flask app init, config, shared utils, blueprint registration
    - ├── blueprints/ # NEW: Subdirectory for blueprint modules
    - │ ├── __init__.py

    - │ ├── orders.py
    - │ ├── suppliers.py
    - │ ├── hpe_mappings.py
    - │ ├── quickbooks.py
    - │ ├── reports.py
    - │ └── utils_routes.py
    - ├── document_generator.py # Service module
    - ├── email_service.py # Service module
    - ├── iif_generator.py # Service module
    - ├── shipping_service.py # Service module
    - ├── Dockerfile
    - └── entrypoint.sh
- **Frontend (order-processing-app-frontend directory):**
    - Framework/Library: React (Vite build tool)
    - Routing: react-router-dom
    - Authentication: Firebase Client SDK
    - Styling: CSS modules, 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).
- **External APIs:** BigCommerce, UPS, FedEx, Postmark.

---

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

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

- **app.py**:
    - Initializes the Flask application (app = Flask(__name__)).
    - Loads environment variables and configurations.
    - Sets up CORS, database engine (engine), Google Cloud Storage client (storage_client), and Firebase Admin SDK.
    - Contains shared utility functions (e.g., convert_row_to_dict, make_json_safe, _get_bc_shipping_address_id, get_hpe_mapping_with_fallback).
    - Defines the verify_firebase_token decorator for API authentication.
    - Imports all blueprint objects from the blueprints directory.
    - Registers all blueprints with the Flask app instance, applying appropriate url_prefix (e.g., app.register_blueprint(orders_bp, url_prefix='/api')).
    - Contains basic routes like / and /test_db.
- **blueprints/orders.py**: Handles API routes related to order management: /api/orders (GET), /api/orders/<id> (GET), /api/orders/status-counts (GET), /api/orders/<id>/status (POST), /api/orders/<id>/process (POST), /api/ingest_orders (POST). Imports shared resources from app.py and relevant service modules.
- **blueprints/suppliers.py**: Handles CRUD API routes for suppliers under /api/suppliers.
- **blueprints/hpe_mappings.py**: Handles CRUD API routes for HPE PO description mappings under /api/hpe-descriptions and lookup routes under /api/lookup/*.
- **blueprints/quickbooks.py**: Handles API routes for QuickBooks integration, including scheduled IIF generation (/api/tasks/*) and on-demand sync (/api/quickbooks/trigger-sync).
- **blueprints/reports.py**: Handles API routes for generating reports, like /api/reports/daily-revenue.
- **blueprints/utils_routes.py**: Handles utility API routes, such as /api/utils/generate_standalone_ups_label.

- **document_generator.py**: Generates PDF documents (POs, Packing Slips) using ReportLab.
- **email_service.py**: Manages sending transactional emails via Postmark API.
- **iif_generator.py**: Generates IIF files for QuickBooks Desktop (POs, Sales Orders, Payments).
- **shipping_service.py**: Manages interactions with UPS and FedEx APIs for label generation and BigCommerce shipment updates.
- **Dockerfile**: Defines the container image for deployment to Cloud Run. Uses python:3.9-slim base image, copies application code, installs dependencies, and uses entrypoint.sh to start the application.
- **entrypoint.sh**: Shell script executed by the container to start the Gunicorn WSGI server. The current version includes verbose Gunicorn logging flags and was successful in deployment:

Bash

```
#!/bin/sh

set -ex

echo "ENTRYPOINT: Script starting."

# ... (other debug echos as per last successful version) ...

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 (order-processing-app-frontend):**
    - Key components like App.jsx, Dashboard.jsx, OrderDetail.jsx, AuthContext.jsx, etc., define the UI and user interactions.
    - OrderDetail.css contains styling for the order detail page.

---

5. Database Schema Notes & Key Tables

(No changes in this session. Refer to Section 5 of Handover Report 20250519b.pdf for details on tables like orders, order_line_items, suppliers, purchase_orders, po_line_items, shipments, hpe_part_mappings, hpe_description_mappings, qb_product_mapping.) The QuickBooks sync status columns (qb_sales_order_sync_status, qb_po_sync_status, etc.) are defined in the schema and are crucial for the IIF generation logic.

---

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

(Refer to Section 6 of Handover Report 20250519b.pdf for base setup. Key updates from this session:)

- **Backend (order-processing-app):**
    - **Virtual Environment (venv):** Absolutely critical.
        - If encountering "Fatal error in launcher" for pip or gunicorn on Windows, the venv is likely corrupted or has incorrect pathing (possibly due to project folder moves or OneDrive interactions). **Solution:** Deactivate, delete the venv folder, and recreate it using python -m venv venv from the project root. Then activate and run python -m pip install -r requirements.txt.
    - **PowerShell Execution Policy (Windows):** If .\venv\Scripts\activate fails with a "scripts disabled" error, run PowerShell as administrator and execute Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser (or LocalMachine).
    - **Gunicorn on Windows:** Gunicorn may raise a ModuleNotFoundError: No module named 'fcntl' when run directly on Windows because fcntl is Unix-specific. This error will NOT occur in the Linux-based Cloud Run container. For local WSGI server testing on Windows, consider using Flask's built-in

server (flask run) or waitress (pip install waitress; waitress-serve --port=8080 app:app).

    - **Line Endings:** Ensure shell scripts like entrypoint.sh are saved with LF (Unix/Linux) line endings, not CRLF (Windows), to prevent execution issues in the Linux container.

- **Frontend (order-processing-app-frontend):** Standard Node.js/npm setup.
- **Deployment:**
    - **Backend (Cloud Run):**
        - Build Docker image: docker build -t YOUR_IMAGE_NAME .
        - Tag: docker tag YOUR_IMAGE_NAME YOUR_REGION-docker.pkg.dev/YOUR_PROJECT_ID/YOUR_REPO_NAME/YOUR_IMAGE_NAME:TAG
        - Push: docker push YOUR_REGION-docker.pkg.dev/YOUR_PROJECT_ID/YOUR_REPO_NAME/YOUR_IMAGE_NAME:TAG
        - Deploy: gcloud run deploy SERVICE_NAME --image ... --region ... (refer to user's provided comprehensive deploy command for all flags).
    - **Frontend (Firebase Hosting):** npm run build, then firebase deploy --only hosting.

---

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

- **Backend Refactoring:**
    - Successfully refactored the monolithic app.py into a main app.py and multiple blueprint files (orders.py, suppliers.py, hpe_mappings.py, quickbooks.py, reports.py, utils_routes.py) located in a blueprints/ subdirectory.
    - Updated import statements within blueprints to use from app import ... for shared objects and import service_module for service modules, resolving previous relative import errors.
- **Deployment Troubleshooting & Resolution:**

    - Addressed local Windows "Fatal error in launcher" for pip and gunicorn by recreating the virtual environment.
    - Resolved PowerShell script execution policy issues for venv activation.
    - Diagnosed and fixed ImportError: attempted relative import beyond top-level package in Cloud Run deployments.
    - Systematically debugged "container failed to start" errors on Cloud Run by:
        - Verifying entrypoint.sh line endings (LF).
        - Testing with a simplified entrypoint.sh to confirm script executability and basic logging.
        - Modifying entrypoint.sh to use verbose Gunicorn logging (--log-level debug, logging to stdout/stderr).
    - Achieved a **successful Cloud Run deployment** with the refactored backend and verbose Gunicorn settings.
- **Frontend (OrderDetail.jsx):**
    - Initial discussion of UI improvements for the OrderDetail.jsx page based on user request.
    - User provided OrderDetail.jsx and OrderDetail.css files.
    - Preliminary analysis and suggestions made for status alignment (mobile) and changing desktop layout from "table" to "cards".

---

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

- **Immediate Post-Handover Tasks & Verification:**
    1. **Thorough API Testing (Critical):**
        - **Action:** Test all API endpoints extensively (GET, POST, PUT, DELETE for all resources) using the frontend or a tool like Postman to ensure the backend refactoring did not introduce regressions.
        - **Goal:** Confirm all backend functionalities behave as expected after the blueprint reorganization.
    2. **Monitor Cloud Run Logs:**

        - **Action:** Keep an eye on the logs for the successfully deployed revision (g1-po-app-backend-service-00212-hwh or the latest successful one) during testing for any runtime errors.
        - **Goal:** Ensure stability and identify any new issues.
    3. **Investigate Firebase auth/quota-exceeded Error:**
        - **Action:** Check Firebase project Authentication/Identity Platform usage quotas in the Google Cloud Console.
        - **Goal:** Understand the cause and resolve it, potentially by upgrading to the Firebase Blaze plan if free tier limits are being consistently hit.
    4. **Review Gunicorn Configuration for Production:**
        - **Action:** The current entrypoint.sh uses --workers 1 --threads 2 --log-level debug. Once stability is confirmed, consider adjusting worker/thread counts for optimal performance and changing --log-level to info for production.
        - **Goal:** Stable and performant production configuration.
- **Frontend UI Enhancements (OrderDetail.jsx):**
    1. **Mobile View - Status Alignment:**
        - **Action:** Implement CSS changes (likely using Flexbox on .order-title-section) to ensure the order status badge is on the same line as the order identifier and right-justified.
        - **Goal:** Improved mobile layout as per user request.
    2. **Desktop View - Fix Duplicate Status:**
        - **Action:** Identify and remove the cause of the duplicate order status display on desktop. This will require careful inspection of OrderDetail.jsx and how status is rendered in different contexts or potentially by overlapping UI elements.
        - **Goal:** Clean and non-redundant status display.
    3. **Desktop View - "Table" to Cards Layout:**
        - **Action:** Clarify with the user which "table" sections they refer to. If it's the overall layout of sections like "Order Information", "Fulfillment

Mode", etc., refactor the JSX and CSS to use a responsive card layout (e.g., using CSS Grid or Flexbox to arrange these <section class="card"> elements in columns on desktop). If it refers to item lists within forms, discuss feasibility and design implications.

    - **Goal:** More modern and consistent UI on desktop, matching the card-based feel of mobile where appropriate.

- **FedEx Integration Finalization (Carry-over from v14.0):**
    - FedEx "Bill RECIPIENT" Sandbox Testing (Blocked, requires FedEx Support).
    - FedEx Production Label Certification (Pending).
    - Integrate FedEx label generation into process_order_route in blueprints/orders.py once certified.
    - (Optional) Standalone FedEx Label Generator UI & Backend.
- **QuickBooks Integration - Full Testing & Refinement (Carry-over from v14.0):**
    - Thoroughly test IIF import for Sales Orders/Payments and Purchase Orders into a TEST QuickBooks Desktop company file. Verify all data points (customer, dates, doc numbers, terms, addresses, line items, amounts, payment application).
    - Refine backend /api/quickbooks/trigger-sync endpoint in blueprints/quickbooks.py for precise database status updates (qb_..._sync_status, qb_..._synced_at, qb_..._last_error) based on returned IDs from iif_generator.py.
    - Enhance frontend QuickbooksSync.jsx to display last sync timestamp, pending counts, and detailed feedback (requires new backend endpoints for this data).
- **MVP Task Completion (Consolidated from v14.0):**
    - PO Export Feature (Excel - Backend & Frontend).
    - Finalize Authentication & Authorization Robustness (error handling, decorator application).
- **Infrastructure & Deployment Finalization (Consolidated from v14.0):**
    - Postmark - External Domain Sending Approval (DKIM, SPF).

    - Cloudflare DNS for Production Domains (if applicable).
- **Long-Term Improvements & Best Practices (Consolidated from v14.0):**
    - Backend Daily Revenue Aggregation Timezone alignment if needed.
    - Security Hardening (Cloud Run Ingress/IAP, input validation, structured logging).
    - Asynchronous Task Processing (Google Cloud Tasks for process_order, batch IIF).
    - Admin User Approval UI (Firebase Custom Claims).
    - QuickBooks Integration - QBWC Exploration (if IIF limitations become problematic).
    - Automated Unit/Integration Tests (Pytest, Jest/React Testing Library).
    - Order Comment Parsing Reliability (consider BigCommerce Metafields).

---

9. Known Issues & Considerations (Updated & Consolidated)

- **Firebase Authentication Quota:**
    - **NEW (Observed this session):** Frontend displaying Error: Firebase: Error (auth/quota-exceeded). Requires investigation of Firebase project Authentication/Identity Platform usage quotas and potential upgrade to the Blaze plan.
- **FedEx Integration Blockers:**
    - "Bill RECIPIENT" Sandbox Testing: Still blocked, awaiting FedEx Developer Support.
    - Production Label Certification: Pending completion.
- **QuickBooks Desktop & IIF Limitations:**
    - IIF is a fragile format; limited error feedback, no updates to existing QB transactions. QBWC or QB Online API are more robust future alternatives if needed.
- **Scalability of Synchronous Processing:**

    - Current API routes like order processing and on-demand QB sync are synchronous. Consider asynchronous tasks for higher volumes.
- **Local Development on Windows with Gunicorn:**
    - Gunicorn may raise ModuleNotFoundError: No module named 'fcntl'. This is Windows-specific and does not affect Linux-based Cloud Run deployments. Use Flask's dev server or Waitress for local WSGI testing on Windows.
- **Environment Management, Quotas, Key Security, Reporting Timezone, BigCommerce API Rate Limits:** As detailed in v14.0 (Section 8).

---

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

The G1 PO App has undergone a significant backend refactoring, moving from a monolithic app.py to a more organized structure using Flask Blueprints. After resolving several deployment challenges related to virtual environments, import paths, and container startup, the application with the refactored backend has been **successfully deployed to Cloud Run** using a verbose Gunicorn configuration in the entrypoint.sh script.

Immediate Priorities for the Incoming Developer:

1. **Thorough End-to-End Testing:**
    - Verify all API endpoints and application functionalities after the backend refactoring to ensure no regressions were introduced.
    - Monitor Cloud Run logs for the current successful revision during testing.
2. **Address Firebase Authentication Quota:**
    - Investigate the auth/quota-exceeded error seen on the frontend. Check Firebase project quotas and consider upgrading the plan if necessary.
3. **Implement OrderDetail.jsx UI Enhancements:**
    - Address the requested changes for mobile (status alignment) and desktop (fix duplicate status, implement card-based layout for main sections).
4. **Review and Optimize Gunicorn Production Settings:**
    - The current deployment uses --workers 1 --threads 2 --log-level debug. Once stability is confirmed, adjust these for production (e.g., potentially more threads, log level to info).

5. **Continue with Pending Major Features:**
    - Resume work on FedEx integration once external blockers are cleared.
    - Proceed with comprehensive testing and refinement of the QuickBooks IIF integration module.

This report, along with the codebase (including the newly structured blueprints directory and updated app.py), previous handover notes, and the entrypoint.sh and Dockerfile, should provide a solid foundation for taking over the project and guiding it to completion.