Project Handover Report: G1 PO App (Version 2 - May 23, 2025)

Date of Report: May 23, 2025 **Compiled By:** Gemini Assistant (based on interactions and provided "Handover Report 20250522b.pdf") **Project:** G1 PO App (Purchase Order and Shipment Automation)

Objective of this Report: To provide a comprehensive overview of the G1 PO App project, including its goals, current state, technology stack, key architectural components, development setup, a summary of recent work, and a detailed plan for future tasks. This document is intended to facilitate a smooth handover to a human developer.

Table of Contents:

---

1. Project Goal & Core Value Proposition (As per original Handover Report)

The primary goal is to develop the "G1 PO App," a secure, internal web application to automate and streamline the purchase order (PO) and shipment processes for both drop-shipped and G1-stocked items. This application aims to minimize manual effort, enhance accuracy, serve as a central platform for critical business workflows, and improve overall operational efficiency. Key functionalities include ingesting orders from BigCommerce, managing orders through a web interface, generating POs, packing slips, and shipping labels (UPS & FedEx), facilitating supplier communication, updating BigCommerce with shipment details, and integrating with QuickBooks Desktop via IIF files.

---

2. Current State of the Application (As of End of May 23, 2025)

The application has a significant number of core features implemented. A major backend refactoring introduced Flask Blueprints for better organization and scalability. The refactored backend has been successfully deployed to Google Cloud Run after resolving various deployment and environment issues. The local development environment has been re-established and critical bugs resolved.

Key Achievements & Functionalities (Incorporating all recent updates):

- **Order Management:**
    - Robust BigCommerce order ingestion, including parsing for third-party shipping accounts and storing billing/shipping cost data.
- **Fulfillment Modes:**
    - Supports Single and Multi-Supplier POs, G1 Onsite Fulfillment, and customer-billed shipments.
- **Document Generation:**
    - PDF generation for Purchase Orders and Packing Slips.
    - Packing slip can be generated in a "blind drop ship" version, omitting company branding and using the customer's billing address as the ship-from on the label. This functionality is controlled via a "Branding" dropdown in OrderDetail.jsx.
    - Packing slip font was changed to "Eloquia". (Note: A non-critical backend warning TTFError: psName=b'\xc3\xb8' contains invalid character 195 related to this font's metadata appears on server startup; PDFs will use a default font locally if this specific font file has issues, but this does not crash the server due to try-except blocks in document_generator.py).
- **Shipping Integration:**
    - **UPS:**
        - Functional "Bill Sender" label generation.
        - "Bill Third Party" label generation is now working correctly after addressing an issue with the PaymentInformation block (specifically changing PaymentDetails to PaymentInformation and ensuring ShipmentCharge is an object, not an array) in shipping_service.py.
    - **FedEx:**

        - OAuth is working. "Bill SENDER" labels have been tested in production (test-marked).
        - "Bill RECIPIENT" functionality is implemented in the code but currently encounters a "SHIPMENT.ACCOUNTNUMBER.UNAUTHORIZED" error from the FedEx API, which requires FedEx support/account configuration to resolve for production accounts.
        - FedEx production API credentials are set up and selected based on the environment.
- **Communication:**
    - Email communications for POs to suppliers via Postmark.
- **BigCommerce Updates:**
    - Integration for order data retrieval and updating order status/shipment details.
- **QuickBooks Integration:**
    - Extensive QuickBooks Desktop IIF file generation for Purchase Orders, Sales Orders (as Invoices), and corresponding Payments. Includes US Central Time conversion and specific formatting requirements. The IIF generator was updated to handle SKUs with underscores for product mapping lookups.
- **Authentication:**
    - Firebase Authentication (Google Sign-In with an isApproved custom claim) is implemented for user access control.
- **Frontend (OrderDetail.jsx):**
    - Features a Dashboard, a detailed OrderDetail.jsx page for order processing, and a Utilities section. Dark mode CSS is implemented.
    - OrderDetail.jsx has been updated for FedEx shipping options, UI for "Bill to Customer's FedEx Account".
    - A "Branding" dropdown (formerly "Blind Drop Ship?" checkbox) with options "Global One Technology Branding" and "Blind Ship (Generic Packing Slip)" is implemented and relocated just above the "Process Order" button.

    - Layout adjustments for the "Shipment Information" section and "Process PO" button centering were made.
    - An Uncaught ReferenceError: currentSelectedShippingOption is not defined in OrderDetail.jsx was addressed.
    - **Profitability Analysis Feature:**
        - A "Profitability Analysis" card is implemented in OrderDetail.jsx.
        - "Items Revenue" is calculated by summing line_item.sale_price * line_item.quantity for all line items.
        - "Total Items Cost" is derived from user inputs in the purchase item forms (for active processing) or from a new actual_cost_of_goods_sold field fetched from the backend for "Processed" orders.
        - The card displays estimated profit and margin.
        - The card's background is conditionally styled: green (60% opacity) for profit, red (60% opacity) for loss.
        - The card's title "Profitability Analysis" is an <h3> for styling consistency.
        - The card is hidden if "G1 Onsite Fulfillment" is selected during active processing.
        - The card is displayed for orders with status "Processed" (unless G1 Onsite).
- **Backend Refactoring:**
    - API route logic was moved from the main app.py into Flask Blueprints for improved modularity and has been deployed.
- **Local Development Environment Restoration:**
    - Fixed frontend .env configuration for VITE_API_BASE_URL.
    - Fixed backend CORS configuration in app.py using an environment variable (ALLOWED_CORS_ORIGIN) for both local (http://localhost:5173) and deployed environments. Cloud Run service needs this env var set to the production frontend URL.

    - Addressed gcloud auth application-default login requirement for backend server's permission to Google Services.
    - Resolved font path issues in document_generator.py by using dynamic relative paths (from /app/fonts).
    - Corrected database table name from purchase_order_line_items to po_line_items in orders.py for both selecting historical costs and inserting new PO line items.
- **Dashboard (Dashboard.jsx):**
    - An infinite loop caused by WorkspaceStatusCounts dependencies was fixed.
    - The "Error fetching status counts: signal is aborted without reason" message in the browser console has been identified as benign in the development environment if the dashboard loads data correctly.

---

3. Technology Stack & Project Layout (Largely as per "Handover Report 20250520b.pdf" and "Handover Report 20250522b.pdf", with updates reflected)

- **Backend (order-processing-app directory):**
    - Language/Framework: Python 3.9+, Flask.
    - WSGI Server: Gunicorn.
    - Key Libraries: SQLAlchemy, google-cloud-sql-connector[pg8000], requests, python-dotenv, ReportLab, Pillow, google-cloud-storage, firebase-admin, postmarker, pytz. **Crucially, Flask-CORS has been added and is used in app.py.**
    - Updated Directory Structure:
        - app.py: Main Flask app, shared resources, blueprint registration, **CORS initialization using ALLOWED_CORS_ORIGIN environment variable.**
        - blueprints/: Contains individual Python files for each blueprint (e.g., orders.py, suppliers.py, quickbooks.py, hpe_mappings.py, reports.py, utils_routes.py).
        - fonts/: Directory containing font files (e.g., eloquia-display-regular.ttf).

        - Service Modules: document_generator.py, email_service.py, iif_generator.py, shipping_service.py.
        - Configuration: .env (for local development: DB credentials, API keys, ALLOWED_CORS_ORIGIN), requirements.txt (includes Flask-CORS), Dockerfile, entrypoint.sh.
- **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, App.css). OrderDetail.css was implicitly modified to accommodate layout changes for the branding dropdown and profit display.
    - Configuration: .env file at the root of this directory for local development, defining VITE_API_BASE_URL and Firebase client SDK keys.
- **Database:**
    - PostgreSQL on Google Cloud SQL.
    - A new column customer_shipping_country_iso2 VARCHAR(2) was added to the orders table.
    - **Important:** The table for purchase order line items is named po_line_items.
- **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 (Incorporating updates from recent changes)

- **Backend (order-processing-app):**
    - app.py: Core application setup, initializes Flask, configurations (including CORS from environment variable ALLOWED_CORS_ORIGIN), database, GCS, Firebase Admin SDK. Defines shared helpers. Registers all blueprints. Contains root routes.
    - blueprints/orders.py:
        - Handles order listing, details, status updates, BigCommerce order ingestion, and the main order processing logic (process_order_route).
        - process_order_route was updated to include FedEx label generation logic, handle a "Blind Drop Ship" flag, and enforce that supplier emails are only sent if all three documents (PO, Packing Slip, Label - if label attempted) are generated.
        - get_order_details function updated to calculate and return actual_cost_of_goods_sold (by summing costs from po_line_items linked to purchase_orders for the given sales order) if the order status is "Processed".
        - All SQL queries referencing purchase order line items should use the table name po_line_items.
    - blueprints/suppliers.py: CRUD operations for suppliers.
    - blueprints/hpe_mappings.py: Manages HPE Part Number to PO Description mappings and SKU lookups.
    - blueprints/quickbooks.py: Endpoints for triggering QuickBooks IIF file generation.
    - blueprints/reports.py: For generating reports like the daily revenue report.
    - blueprints/utils_routes.py: Utility endpoints, e.g., standalone UPS label generator.
    - shipping_service.py:
        - Contains logic for interacting with UPS and FedEx APIs to get shipping rates (future) and generate shipping labels. Handles OAuth for both. FedEx logic selects production credentials based on FEDEX_API_ENVIRONMENT.

        - generate_ups_label_raw function was modified to correctly implement "Bill Third Party" functionality (changed PaymentDetails key to PaymentInformation and ensured ShipmentCharge is an object).
    - document_generator.py:
        - Generates PDF documents (Purchase Orders, Packing Slips). Updated to accept is_blind_slip and custom_ship_from_address parameters. Packing slip font changed to "Eloquia".
        - **Critical Fix:** Font paths are now dynamically generated relative to the file's location (os.path.join(os.path.dirname(os.path.abspath(__file__)), 'fonts', 'font_name.ttf')) to work in both local (Windows/Mac/Linux) and Docker environments. The previous hardcoded /app/fonts path caused startup errors locally.
    - email_service.py: Handles sending emails (e.g., POs to suppliers) via Postmark.
    - iif_generator.py: Contains all logic for generating IIF files for QuickBooks Desktop. Updated to handle SKUs with underscores.
    - Dockerfile & entrypoint.sh: Configure the Docker container and how Gunicorn runs the Flask app in Cloud Run. requirements.txt now includes Flask-CORS.
- **Frontend (order-processing-app-frontend):**
    - OrderDetail.jsx:
        - Primary component for viewing and processing individual orders.
        - Significant changes for FedEx shipping methods, UI for "Bill to Customer's FedEx Account," and the "Branding" dropdown (options: "Global One Technology Branding" / "Blind Ship (Generic Packing Slip)").
        - Branding dropdown relocated to be the last input before the "Process Order" button.
        - Layout adjustments for "Shipment Information" and "Process PO" button centering.

        - Fixes for the "page goes blank" error related to currentSelectedShippingOption.
        - **Profitability Analysis Feature:**
            - Displays "Items Revenue" (sum of line items' sale price * quantity), "Total Items Cost" (from form inputs or historical data for processed orders), "Est. Items Profit," and "Est. Items Margin."
            - Card background is green (60% opacity) for profit, red (60% opacity) for loss.
            - Title is <h3>Profitability Analysis</h3>.
            - Hidden for "G1 Onsite Fulfillment" mode when processing.
            - Displayed for orders with status "Processed" (uses actual_cost_of_goods_sold from backend; defaults to $0 cost with a console warning if backend field is missing).
    - Dashboard.jsx:
        - Main dashboard view.
        - An infinite loop issue related to WorkspaceStatusCounts dependencies was resolved by removing errorOrders from its useCallback dependency array.
        - The "Error fetching status counts: signal is aborted without reason" message in the browser console is understood to be a benign development-mode artifact if the counts load correctly.
    - OrderDetail.css: Styles for OrderDetail.jsx, updated for dark mode and new layout adjustments, including the relocated branding dropdown and profit display card.
    - App.css: Global styles, including dark mode variables.
    - AuthContext.jsx: Manages Firebase authentication state and provides apiService for making authenticated API calls.
    - App.jsx: Main application component defining routes and structure.

---

5. Database Schema Notes & Key Tables (Refer to "Handover Report 20250520b.pdf," Section 5, pages 13-15 for the original detailed schema)

- **orders table:**
    - Added customer_shipping_country_iso2 VARCHAR(2): Stores the 2-letter ISO code for the shipping country to resolve an ingestion error.
    - The customer_shipping_country column should store the full country name for display purposes.
    - QuickBooks sync status columns (qb_po_sync_status, qb_sales_order_sync_status, etc.) are crucial for IIF generation logic.
    - Ensure customer_ups_account_zipcode is being correctly populated during ingestion if it's intended to be used for third-party UPS billing.
    - The backend now adds a temporary field actual_cost_of_goods_sold to the order object sent to the frontend *if* the order status is "Processed". This is calculated on-the-fly by summing costs from related po_line_items and is **not a new column** in the orders table.
- **po_line_items table:** (Previously referred to inconsistently as purchase_order_line_items in some backend queries)
    - **Correct Name:** The actual table name used for storing purchase order line items is po_line_items. All backend SQL queries (SELECTs, INSERTs) have been updated to use this correct name.
    - Contains purchase_order_id, unit_cost, quantity, etc., which are used to calculate actual_cost_of_goods_sold for processed orders.
- **purchase_orders table:**
    - Contains order_id (foreign key to orders.id) linking POs to the main sales order.

---

6. Developer Setup & Environment (Local & Cloud) (Refer to "Handover Report 20250520b.pdf," Section 6, pages 16-18 for base setup)

- **Backend (order-processing-app directory):**
    - Python 3.9+ virtual environment is essential (e.g., python -m venv venv, then activate).

    - Install dependencies: pip install -r requirements.txt (ensure Flask-CORS is in this file).
    - PowerShell Execution Policy (Windows): May need Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser.
    - Gunicorn on Windows: Use Flask's dev server (flask run) or Waitress for local WSGI testing due to fcntl error.
    - Line Endings: Ensure .sh files use LF (Unix) line endings.
    - **Local .env file (in order-processing-app/):**
        - Must contain all necessary database credentials (DB_CONNECTION_NAME, DB_USER, DB_PASSWORD, DB_NAME), BigCommerce keys, UPS/FedEx/Postmark API keys, GCS bucket names, Ship From details.
        - **Crucially, add:** ALLOWED_CORS_ORIGIN="http://localhost:5173" (or whatever port the local frontend runs on).
    - Run local server: flask run --port=8080 (or your preferred port).
- **Frontend (order-processing-app-frontend directory):**
    - Node.js and npm required.
    - Install dependencies: npm install.
    - **Local .env file (in order-processing-app-frontend/):**
        - Must contain:
            - VITE_API_BASE_URL="http://localhost:8080/api" (ensure port matches the local backend).
            - All Firebase SDK configuration keys (VITE_FIREBASE_API_KEY, etc.).
    - Run local server: npm run dev (usually on port 5173).
- **Authentication for Local Backend:**
    - The backend server (when running locally) needs Application Default Credentials (ADC) to communicate with Google services (like Firebase Admin SDK for token verification, GCS).

    - Run gcloud auth application-default login in the terminal where the Flask app will run, and authenticate with a Google account that has access to the Firebase/GCP project. These credentials expire periodically and may need to be refreshed.

- **Deployment:**
    - **Backend to Cloud Run (via Docker):**
        - Ensure Flask-CORS is in requirements.txt.
        - The Dockerfile and entrypoint.sh should correctly build and run the Flask/Gunicorn app.
        - **Crucial Cloud Run Environment Variable:** Set ALLOWED_CORS_ORIGIN to the deployed Firebase Hosting URL (e.g., https://g1-po-app-77790.web.app). All other secrets (DB credentials, API keys) should be set as environment variables or secrets in the Cloud Run service configuration.
    - **Frontend to Firebase Hosting:**
        - Run npm run build.
        - Deploy using firebase deploy --only hosting.
        - Ensure Firebase Hosting is configured to serve the dist directory.
        - Environment variables like VITE_API_BASE_URL for the *deployed* frontend are typically set during the build process or via Firebase Hosting configuration if it supports injecting them (less common for static sites; usually baked in at build from the build environment's vars or a .env.production file if used). For this project, VITE_API_BASE_URL is likely set in the build pipeline or needs to point to the Cloud Run service URL.
- **Font Paths:** document_generator.py now uses dynamic relative paths for fonts, which should work seamlessly in both local and Docker environments, assuming the fonts directory is correctly placed relative to document_generator.py and included in the Docker image.

---

7. Summary of Interactions (May 21-22, 2025 - per original PDF) This section summarizes work completed prior to the most recent interactive sessions, as documented in the "Handover Report 20250522.pdf". Key items included:

- FedEx Integration Advancement ("Bill Recipient" logic review, UI updates, API error diagnosis).
- QuickBooks IIF Generator Update (handling SKUs with underscores).
- Order Processing Enhancements (strict document check before supplier email).
- Initial Blind Drop Ship Functionality (frontend checkbox, backend logic, document generator updates).
- Font Change to "Eloquia" in document_generator.py.
- Frontend Layout Adjustments (OrderDetail.jsx & OrderDetail.css).
- Database Schema Update (customer_shipping_country_iso2 added to orders).
- Critical Bug Fix (currentSelectedShippingOption in OrderDetail.jsx).

---

8. Summary of Interactions (Latter Part of May 22-23, 2025 - Current Chat Session) This section details the work completed during the recent interactive sessions, building upon the state described in the original PDF:

- **UPS "Bill Third Party" Shipping Fix:**
    - Resolved failures by correcting the PaymentInformation block (key name and ShipmentCharge structure) in shipping_service.py to align with UPS API v2409 specifications. Confirmed working by the user.
- **OrderDetail.jsx UI Enhancements (Branding Dropdown):**
    - Labels for the branding dropdown options were updated to "Global One Technology Branding" and "Blind Ship (Generic Packing Slip)".
    - The "Branding" dropdown was relocated to be the last input just above the "Process Order" button in both single-supplier and multi-supplier views.
- **Local Development Environment Troubleshooting & Setup:**
    - Configured frontend .env file with VITE_API_BASE_URL to point to the local backend.

    - Implemented backend CORS handling using Flask-CORS in app.py, controlled by an ALLOWED_CORS_ORIGIN environment variable (set in backend .env for local, and to be set in Cloud Run for production).
    - Addressed Firebase Admin SDK authentication issues for the local backend by running gcloud auth application-default login.
    - Resolved critical font path errors in document_generator.py by changing hardcoded /app/fonts paths to dynamic, relative paths.
    - Corrected database table name from purchase_order_line_items to po_line_items in orders.py (in get_order_details for fetching historical costs, and in process_order_route for inserting new PO line items).
- **Dashboard.jsx Stability:**
    - Fixed an infinite loop caused by incorrect dependencies in the useCallback for WorkspaceStatusCounts.
    - Clarified that the "signal is aborted without reason" browser console message is often benign in development.
- **Profitability Analysis Feature (OrderDetail.jsx):**
    - Implemented and debugged the display logic.
    - Corrected the revenue data source from orderData.order.total_inc_tax (which was undefined) to orderData.order.total_sale_price, and subsequently refined it to sum line_item.sale_price * line_item.quantity for "Items Revenue".
    - Added conditional background color to the card (green for profit, red for loss, with 60% opacity).
    - Changed the card's title to an <h3> for styling consistency.
    - Ensured the card is hidden when "G1 Onsite Fulfillment" mode is selected (during active processing).
    - Enabled the card to display for orders with status "Processed". This involved:
        - Updating the get_order_details function in blueprints/orders.py to calculate and return actual_cost_of_goods_sold (sum of related po_line_items costs) when an order is "Processed".

- Updating the profit calculation useEffect in OrderDetail.jsx to use this actual_cost_of_goods_sold field for "Processed" orders (defaulting to $0 cost if the field is missing, with a console warning).

---

9. Detailed Plan for Completion & Future Tasks This plan integrates tasks from the previous "Handover Report 20250522b.pdf" with progress from all recent sessions and considerations for future development.

Phase 0: Immediate Post-Handover Validation & Critical Fixes

1. **Thorough End-to-End Testing (Critical):**
    - **Action:** Systematically test all API endpoints and frontend workflows.
    - **Focus:**
        - Confirm the backend refactoring into blueprints and all recent fixes operate as expected without regressions. This includes:
            - FedEx "Bill Recipient" (once FedEx account configuration is resolved for the "UNAUTHORIZED" error).
            - UPS "Bill Third Party" (should be fully working).
            - Blind Drop Ship functionality with correct document generation and ship-from address changes.
            - QuickBooks IIF generation with SKU underscore handling.
            - Strict document check before supplier email dispatch in process_order_route.
            - Correct usage of po_line_items table name throughout the backend.
        - Verify data handling, all document generation types (PO, Packing Slip regular & blind), and all external API calls (BigCommerce, Postmark, UPS, FedEx).
        - Test the OrderDetail.jsx branding dropdown changes and their impact on data passed to process_order_route.
        - **Verify Profitability Analysis:**

        - Confirm it appears correctly for active processing (Single/Multi-Supplier, not G1 Onsite).
        - Confirm it appears for "Processed" orders (not G1 Onsite).
        - Confirm revenue is sum of line items.
        - Confirm cost is from form inputs for active processing.
        - **Critically test that actual_cost_of_goods_sold is correctly calculated by the backend and used by the frontend for "Processed" orders.**
        - Verify conditional background colors and h3 title styling.
    - **Goal:** Confirm stability, data integrity, and correct operation of all recent fixes and features.
2. **Cloud Deployment Checks:**
    - **Action:** After deploying the latest backend code to Cloud Run, verify the ALLOWED_CORS_ORIGIN environment variable is correctly set to the production frontend URL (e.g., https://g1-po-app-77790.web.app).
    - Monitor Cloud Run logs during testing for any new runtime errors or unexpected behavior.
    - **Goal:** Stable and correctly configured cloud deployment.
3. **Resolve Firebase auth/quota-exceeded Error (Verify Fixed):**
    - **Action:** The user has likely upgraded their Firebase project to the "Blaze" plan. Confirm this is still active and no quota issues are resurfacing.
    - **Goal:** Reliable Firebase Authentication.
4. **Review & Tune Gunicorn Production Settings:**
    - **Action:** The current entrypoint.sh might use debug settings. After testing confirms stability, adjust Gunicorn for production (e.g., increase threads/workers, set log-level to info or warning).
    - **Goal:** Optimized and stable Gunicorn configuration for Cloud Run.
5. **Frontend Bug Squashing:**

- **Action:** Thoroughly test all supplier/mode selections in OrderDetail.jsx to ensure no "blank page" errors occur and that the profit display logic and branding dropdown logic work correctly across all modes and order statuses. Check browser console for any other JS errors.
- **Goal:** Stable OrderDetail.jsx component during dynamic state changes.

6. **Address Font Metadata Warning (Low Priority):**
    - **Action:** Investigate the TTFError: psName=b'\xc3\xb8' contains invalid character 195 for the Eloquia font. This may require finding a "cleaner" version of the font file or using a font editor to remove/correct the problematic PostScript name metadata.
    - **Goal:** Eliminate the non-critical startup warning in the backend logs and ensure custom fonts render correctly in PDFs.

Phase 1: Frontend UI/UX Refinement (Some items from original report, potentially impacted by recent changes)

1. **Finalize Shipment Information Layout (Desktop & Mobile):**
    - Action: Review and implement the latest discussed CSS and JSX for the "Method", "Weight (lbs)" fields to achieve the desired layout in both Single Supplier and Multi-Supplier (per PO) views.
    - Desktop: "Method" on its own row. "Weight (lbs)" label + input on the next row. (Note: The original report mentioned "Blind Ship?" here, which has now been moved and redesigned as "Branding" dropdown).
    - Mobile: Ensure these elements stack cleanly and logically.
    - Goal: User-approved layout for shipment inputs.
2. **Item Purchase Grid Layout - "Unit Cost" (Desktop - Single Supplier View):**
    - Action: Adjust CSS for .item-row or .qty-cost-row within the "Items to Purchase" section to ensure the "Unit Cost" input field appears directly under its "Unit Cost" label, similar to how "Qty" is structured.
    - Goal: Consistent and clear layout for item entry.
3. **Mobile View - Status Alignment:**
    - Action: Verify the order status badge aligns correctly with the order title on mobile views. Adjust CSS if necessary.

- Goal: Improved mobile layout.

4. **Desktop View - Address Duplicate Status / Layout Transformation:**
    - Action: Clarify with the user which "table" layouts need conversion to cards and where any duplicate statuses appear. Implement a responsive multi-column card layout.
    - Goal: Modernized and responsive UI for list views.
5. **Styling for Relocated Branding Dropdown:**
    - Action: Ensure the relocated "Branding" dropdown (and its "Global Options" container in multi-supplier mode) is styled correctly and consistently with the rest of the form, especially concerning label alignment and spacing. Adjust OrderDetail.css as needed.
    - Goal: Visually integrated and clean UI for the branding options.

Phase 2: Core Functionality Enhancements & New Features

1. **UPS International Shipping:**
    - Action: Research UPS API requirements for international shipments (customs forms, commodity info, etc.). Update shipping_service.py and OrderDetail.jsx for new fields and UI elements. Update document_generator.py if needed for local forms. Consider database changes for customs-related product info.
    - Goal: Enable UPS international shipping label and document generation.
2. **FedEx "Bill RECIPIENT" Authorization Fix:**
    - Action: Follow up with FedEx support to resolve the "SHIPMENT.ACCOUNTNUMBER.UNAUTHORIZED" error and configure accounts appropriately.
    - Goal: Fully functional FedEx "Bill RECIPIENT" label generation.
3. **Shipping Rate Estimates:**
    - Action: Implement rate fetching from UPS/FedEx in shipping_service.py. Update OrderDetail.jsx to display rates.
    - Goal: Provide users with shipping cost estimates.
4. **Address Validation:**

- Action: Integrate UPS/FedEx address validation APIs into shipping_service.py. Provide feedback in OrderDetail.jsx.
- Goal: Reduce shipping errors.

5. **Enhanced Error Handling & User Feedback:**
   - Action: Review all external API interactions for more robust error handling and clearer frontend feedback.
   - Goal: Improve application resilience and user experience.
6. **Automated Tests (Backend & Frontend):**
   - Action: Develop unit and integration tests.
   - Goal: Ensure code quality and prevent regressions.
7. **Consolidate Frontend State Management (If needed):**
   - Action: Evaluate if a more robust state management solution (e.g., Redux Toolkit, Zustand) is needed beyond useState/useCallback.
   - Goal: Maintainable and scalable frontend architecture.
8. **Comprehensive User Documentation/Guide:**
   - Action: Create a user guide for all features.
   - Goal: Enable effective application use.

Phase 3: Advanced Features & Integrations

1. **Partial Shipment Tracking within a Single Order:**
   - Action: Enhance UI to clearly track each part of a multi-PO/shipment order individually.
   - Goal: Better visibility for complex orders.
2. **Inventory Management Aspects (Future Consideration):**
   - Action: Explore basic inventory tracking for G1-stocked items.
   - Goal: Potential expansion of app capabilities.
3. **More Sophisticated Reporting:**

- Action: Expand on the current daily revenue report with more customizable and detailed reporting.
- Goal: Provide better business intelligence.

This detailed plan should provide a clear roadmap for the continued development and completion of the G1 PO App. It's crucial to test existing functionality thoroughly after the recent environment and logic fixes before moving heavily into new features.