* + 1. o the configured GCS bucket under a structured path (processed\_orders/order\_<id>/...).
    2. **Update DB with GCS Paths:** Updates the purchase\_orders record with po\_pdf\_gcs\_path and the shipments record with packing\_slip\_gcs\_path and label\_gcs\_path.
    3. **Email Supplier:** Calls email\_service.send\_po\_email, passing supplier email, PO number, and the PDF bytes for PO, Packing Slip, and Label. Uses Postmark API.
    4. **Update BigCommerce:** Calls shipping\_service.update\_bigcommerce\_order to create a shipment in BigCommerce using the tracking number, line item details (bigcommerce\_line\_item\_id), and the fetched BigCommerce order\_address\_id. This implicitly updates the BC order status.
    5. **Update Local Order Status:** Updates the status of the order in the local orders table to 'Processed'.
    6. **Commit/Rollback:** Commits the transaction upon successful completion of all steps; rolls back on failure.
* **Service Modules:**
  + shipping\_service.py: Contains functions for UPS OAuth, UPS label generation (including state mapping), and BigCommerce order updates. Tested successfully.
  + document\_generator.py: Contains functions to generate PO and Packing Slip PDFs using ReportLab. Tested successfully within the process\_order flow, but argument signatures were determined iteratively via TypeError messages. Needs review for expected arguments and use of OptionPN vs original SKU.
  + email\_service.py: Contains send\_po\_email function using Postmark API. Tested successfully for sending to internal domain (Postmark account approval pending for external domains).
* **Frontend (React/Vite):**
  + Basic project structure exists (main.jsx, App.jsx).
  + Routing set up using react-router-dom.
  + Placeholder components created (Dashboard.jsx, OrderDetail.jsx, Suppliers.jsx, Products.jsx).
  + Dashboard.jsx fetches and displays orders (currently unfiltered, needs update to filter by status='new').
  + Navigation from Dashboard to OrderDetail page implemented.
  + **Significant development required on OrderDetail.jsx** to implement the form for triggering the processing workflow.

**4. Technology Stack**

* **Backend:** Python 3, Flask, SQLAlchemy (Core API with text()), pg8000 (PostgreSQL driver)
* **Database:** PostgreSQL (hosted on Google Cloud SQL)
* **Cloud:** Google Cloud Platform (GCP) - Cloud SQL, Google Cloud Storage (GCS), Secret Manager (planned), Cloud Run (planned)
* **APIs:**
  + UPS Shipping API (JSON REST, v2409 used in testing, CIE test env)
  + BigCommerce API (V2 REST)
  + Postmark API (Email)
* **Document Generation:** ReportLab
* **Frontend:** React, Vite, react-router-dom
* **Environment:** Python venv, .env file for configuration, requirements.txt for dependencies.

**5. Codebase Structure**

* **app.py:** Main Flask application. Initializes DB engine, GCS client. Defines all API routes (/, /test\_db, /api/orders, /api/suppliers, /api/products, /ingest\_orders, /api/orders/<id>/process). Contains orchestration logic for /ingest\_orders and /api/orders/<id>/process. Includes helper functions (convert\_row\_to\_dict, make\_json\_safe, \_get\_bc\_shipping\_address\_id).
* **shipping\_service.py:** Contains helper functions for UPS API interactions (get\_ups\_oauth\_token, map\_shipping\_method\_to\_ups\_code, generate\_ups\_label) and BigCommerce API interaction (update\_bigcommerce\_order).
* **email\_service.py:** Contains send\_po\_email function using Postmark.
* **document\_generator.py:** Contains generate\_purchase\_order\_pdf and generate\_packing\_slip\_pdf functions using ReportLab. **(Function signatures need verification based on iterative debugging)**.
* **src/ (Frontend):** Standard Vite React project structure. Key components are in src/components/.
* **.env:** Stores all necessary environment variables (Database connection string components, API keys/tokens/secrets for BigCommerce/UPS/Postmark, GCS bucket name, Ship From address details). **This file is critical and must not be committed to version control.**
* **requirements.txt:** Lists all Python dependencies and their versions. Generated using pip freeze.

**6. Database Schema Overview**

*(Assumes tables exist in the Cloud SQL PostgreSQL instance with columns added during development)*

* **orders:** Stores ingested order details. Key columns: id (PK, SERIAL), bigcommerce\_order\_id (INTEGER, UNIQUE), customer\_name, shipping address fields, order\_date (TIMESTAMPTZ), total\_sale\_price (DECIMAL), status (VARCHAR, e.g., 'new', 'Processed', 'international\_manual'), is\_international (BOOLEAN), created\_at, updated\_at (TIMESTAMPTZ).
* **order\_line\_items:** Stores original line items from BigCommerce. Key columns: id (PK, SERIAL), order\_id (FK to [orders.id](http://orders.id/)), bigcommerce\_line\_item\_id (INTEGER), sku (VARCHAR - **This is the original BigCommerce SKU**), name (TEXT), quantity (INTEGER), sale\_price (DECIMAL), created\_at, updated\_at (TIMESTAMPTZ).
* **suppliers:** Stores supplier details. Key columns: id (PK, SERIAL), name (VARCHAR), email (VARCHAR, UNIQUE), address\_line1, etc., payment\_terms (VARCHAR), created\_at, updated\_at (TIMESTAMPTZ).
* **purchase\_orders:** Stores generated POs. Key columns: id (PK, SERIAL), po\_number (INTEGER, UNIQUE NOT NULL), order\_id (FK to [orders.id](http://orders.id/)), supplier\_id (FK to [suppliers.id](http://suppliers.id/)), po\_date (TIMESTAMPTZ), payment\_instructions (TEXT), status (VARCHAR NOT NULL DEFAULT 'New'), total\_amount (DECIMAL), po\_pdf\_gcs\_path (VARCHAR(512) NULL), created\_at, updated\_at (TIMESTAMPTZ).
* **po\_line\_items:** Stores items on a specific PO. Key columns: id (PK, SERIAL), purchase\_order\_id (FK to [purchase\_orders.id](http://purchase_orders.id/)), original\_order\_line\_item\_id (FK to [order\_line\_items.id](http://order_line_items.id/), NULLABLE), sku (VARCHAR - **Should store OptionPN**), description (TEXT), quantity (INTEGER), unit\_cost (DECIMAL), condition (VARCHAR), created\_at, updated\_at (TIMESTAMPTZ).
* **shipments:** Stores shipment details. Key columns: id (PK, SERIAL), purchase\_order\_id (FK to [purchase\_orders.id](http://purchase_orders.id/)), tracking\_number (VARCHAR, UNIQUE NOT NULL), shipping\_method\_name (VARCHAR(255) NULL), weight\_lbs (DECIMAL or FLOAT), label\_gcs\_path (VARCHAR(512) NULL), packing\_slip\_gcs\_path (VARCHAR(512) NULL), created\_at (TIMESTAMPTZ).
* **products:** Basic table for product info. Key columns: id (PK, SERIAL), sku (VARCHAR, UNIQUE), standard\_description (TEXT), created\_at, updated\_at (TIMESTAMPTZ). (May be superseded or augmented by hpe\_part\_mappings).
* **hpe\_part\_mappings:** **(CRITICAL - Needs Creation & Data Migration)** This table is essential for mapping the sku from order\_line\_items to the correct HPE option\_pn required for purchase orders.
  + **Required Columns:** sku (VARCHAR(100) PRIMARY KEY or UNIQUE NOT NULL), option\_pn (VARCHAR(100) NOT NULL), pn\_type (VARCHAR(50) NULLABLE).
  + **Action:** Create this table in PostgreSQL and migrate the ~8000+ rows from the existing MS Access table.

**7. API Integrations & Credentials (.env variables)**

* **BigCommerce:** BIGCOMMERCE\_STORE\_HASH, BIGCOMMERCE\_ACCESS\_TOKEN, BC\_PROCESSING\_STATUS\_ID, BC\_SHIPPED\_STATUS\_ID. Uses V2 REST API.
* **UPS:** UPS\_CLIENT\_ID, UPS\_CLIENT\_SECRET, UPS\_BILLING\_ACCOUNT\_NUMBER. Uses JSON REST Shipping API (v2409 tested) via CIE test environment. *Note: Test env returns duplicate tracking numbers.*
* **Postmark:** EMAIL\_API\_KEY (Server Token), EMAIL\_SENDER\_ADDRESS (Verified Signature), EMAIL\_BCC\_ADDRESS. *Note: Account requires approval to send outside sender domain.*
* **Google Cloud Storage:** GCS\_BUCKET\_NAME. Requires IAM permissions for credentials used by app.py (ADC locally, service account in Cloud Run). Bucket name used in testing: g1-po-app-documents.
* **Database:** DB\_CONNECTION\_NAME (e.g., project:region:instance), DB\_USER, DB\_PASSWORD, DB\_NAME, DB\_DRIVER (pg8000).
* **Ship From Address:** SHIP\_FROM\_NAME, SHIP\_FROM\_CONTACT, SHIP\_FROM\_STREET1, SHIP\_FROM\_STREET2, SHIP\_FROM\_CITY, SHIP\_FROM\_STATE, SHIP\_FROM\_ZIP, SHIP\_FROM\_COUNTRY, SHIP\_FROM\_PHONE.

**8. Environment Setup**

* Python 3.x virtual environment (venv).
* Dependencies installed via pip install -r requirements.txt.
* .env file populated with all necessary credentials (see section 7).
* **Cloud SQL Auth Proxy:** Must be running locally to connect to the Cloud SQL database during development.
* **GCP Project:** Active project with Cloud SQL (PostgreSQL) instance created and running, GCS bucket created.

**9. Detailed Progress Log (Summary of process\_order steps implemented)**

The POST /api/orders/<order\_id>/process endpoint successfully integrates the following sequential steps within a single database transaction:

* **Step 1:** Receive JSON payload (supplier\_id, po\_line\_items, weight, payment\_instructions).
* **Step 2:** Connect to DB and begin transaction.
* **Step 3:** Fetch local order, line items, and supplier data.
* **Step 4:** Validate order is domestic.
* **Step 5:** Generate sequential numeric PO number (starting 200001).
* **Step 6:** Insert record into purchase\_orders table.
* **Step 7:** Insert record(s) into po\_line\_items table (currently uses original SKU - **needs update**).
* **Step 8:** Generate PO PDF and Packing Slip PDF bytes using document\_generator.
* **Step 9:** Generate UPS Label PDF bytes and tracking number using shipping\_service.
* **Step 10:** Insert record into shipments table.
* **Step 11:** Upload PO, Packing Slip, and Label PDFs to GCS; Update purchase\_orders and shipments tables with GCS URIs.
* **Step 12:** Send email with PO, Packing Slip, and Label PDF attachments to supplier via Postmark using email\_service.
* **Step 13:** Update BigCommerce order status and add tracking number using shipping\_service.
* **Step 14:** Update local orders table status to 'Processed'.
* **Commit/Rollback:** Commit transaction on success, rollback on any error.

**10. Critical Next Step: HPE Part Number Mapping**

* **Requirement:** The business logic requires using specific HPE "Option PNs" on Purchase Orders sent to suppliers, rather than the potentially different SKUs used in BigCommerce. A mapping table from the BigCommerce SKU (stored in order\_line\_items.sku) to the required OptionPN exists in an MS Access database (~8000 rows) with columns SKU, OptionPN, PNType.
* **Impact:** This mapping is essential for generating correct POs (both the data stored in po\_line\_items and the generated PO PDF) and for displaying the correct part number information in the UI.
* **Action Plan:**
  1. **DB Schema:** Create the hpe\_part\_mappings table in PostgreSQL (Columns: sku VARCHAR PK/UNIQUE, option\_pn VARCHAR NOT NULL, pn\_type VARCHAR).
  2. **Data Migration:** Export data from MS Access (CSV recommended) and import into the PostgreSQL hpe\_part\_mappings table. Verify counts and data integrity.
  3. **Backend Update (GET /api/orders/<id>):** Modify the SQL query in the get\_order\_details function (app.py) to LEFT JOIN hpe\_part\_mappings on oli.sku = hpm.sku. Return sku, option\_pn, pn\_type etc., for each line item.
  4. **Backend Update (POST /api/orders/<id>/process):**
     + **Step 3 (Fetch Data):** Modify the query fetching local\_order\_line\_items\_list to include the LEFT JOIN to hpe\_part\_mappings as done for the GET endpoint.
     + **Step 7 (Insert PO Line Items):** In the loop, use the option\_pn (retrieved via the join in Step 3) as the value for the sku column when inserting into po\_line\_items. Handle cases where option\_pn is NULL (use original sku as fallback or raise error). Update the po\_items\_for\_pdf list accordingly.
     + **Step 8 (Generate PO PDF):** Ensure the po\_items key passed to generate\_purchase\_order\_pdf uses the list containing OptionPNs.
  5. **Test Backend:** Thoroughly test GET /api/orders/<id> and POST /api/orders/<id>/process to ensure the correct OptionPN is retrieved, used for PO creation (DB and PDF), and returned by the GET endpoint. Test cases should include items with and without mappings.

**11. Remaining MVP Tasks (Post-HPE Mapping)**

*(Based on Project Outline and current status)*

1. **Frontend UI (OrderDetail.jsx):**
   * **Task 11.1:** Display Order Details: Show all relevant info from GET /api/orders/<id>, including line items with original SKU and the retrieved option\_pn clearly differentiated.
   * **Task 11.2:** Build Processing Form: Create form sections for:
     + Supplier Selection: Fetch suppliers using GET /api/suppliers and populate a dropdown.
     + PO Line Items Review/Entry: Display original items with OptionPN. Allow user to confirm quantities. Decide if cost/condition need to be editable here or are fixed based on sourcing rules (for MVP, likely fixed based on input). **The data sent back to the /process endpoint must reference the original line item (original\_order\_line\_item\_id or original sku) so the backend can perform the OptionPN lookup.**
     + Total Shipment Weight input.
     + Payment Instructions input (textarea).
   * **Task 11.3:** Implement "Process Shipment" Button: Gather form data, construct JSON payload, call POST /api/orders/<id>/process.
   * **Task 11.4:** Handle API Response: Display success message (including PO#, Tracking#) or detailed error messages returned from the backend API. Update UI state.
2. **Frontend UI (Dashboard.jsx):**
   * **Task 12.1:** Filter Orders: Update the fetch call to reliably filter orders by status (e.g., status=new).
   * **Task 12.2:** Add PO Export Trigger: Add a button/link to call GET /api/exports/pos.
3. **Backend PO Export (GET /api/exports/pos):**
   * **Task 13.1:** Create Endpoint & Query: Define route, write SQL query joining relevant tables (purchase\_orders, po\_line\_items, suppliers, orders).
   * **Task 13.2:** Generate Excel: Use openpyxl to create and populate the .xlsx file in memory.
   * **Task 13.3:** Return File: Use Flask's send\_file to return the generated Excel file.
4. **Deployment (GCP):**
   * **Task 14.1:** Containerization (Dockerfile for backend, frontend build or Dockerfile).
   * **Task 14.2:** Secret Manager Setup.
   * **Task 14.3:** Cloud Run Service(s) Configuration (DB connection, secrets, IAM roles).
   * **Task 14.4:** Networking (if needed).
   * **Task 14.5:** Deploy and test in cloud environment.
5. **Testing:**
   * **Task 15.1:** End-to-End Testing (UI -> Backend -> APIs -> DB).
   * **Task 15.2:** Edge Case Testing (API errors, missing data, validation, missing HPE mappings).
   * **Task 15.3:** Unit/Integration Tests (Optional but Recommended).
6. **Postmark Account Approval:** Ensure the Postmark account is fully approved for sending to external domains before full production deployment.

**12. Future Enhancements (Post-MVP)**

* Handling International Orders (customs forms, different shipping logic).
* Handling Orders with Multiple Suppliers / Split Shipments.
* QuickBooks Online Integration (instead of Desktop).
* More sophisticated Product/SKU Management UI (including managing HPE mappings).
* User Authentication and Authorization.
* Caching UPS OAuth tokens.
* More robust error handling and user feedback mechanisms.
* Dashboard improvements (more filters, summaries, search).
* Automated testing suite (CI/CD pipeline).

This detailed report should equip the incoming developer with the necessary context and a clear path forward to complete the MVP, starting with the critical HPE part number mapping integration.