- 
    - 
        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 t o 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), 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), supplier_id (FK to 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), original_order_line_item_id (FK to 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), 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.