G1 PO App - Developer Handover Report (Expanded)

Last Updated: June 3, 2025 (reflecting recent development session)

Original Handover Date: May 31, 2025

1. Project Overview

The "G1 PO App" is a web application designed to streamline order processing for Global One Technology. It ingests orders from BigCommerce, allows users to manage these orders, generate purchase orders (POs) to suppliers, create shipping labels (UPS and FedEx), generate packing slips, and manage related international shipment documentation. More recently, functionality has been added for managing HPE Product Descriptions, Customs Product Information, and for generating and emailing customer-facing "Paid Invoices" (Receipts) and "Unpaid Invoices" (specifically for Wire Transfers), including updating order statuses in BigCommerce and the local application. The application features a Flask backend (Python) and a React frontend (Vite). Authentication is handled by Firebase.

2. Project Layout & Core Technologies

This section incorporates details from the original handover report and adds new components.

Frontend:

- **Framework:** React (using Vite for build tooling)
- **Key Page/Component Views:**
    - Dashboard.jsx: Displays orders with filtering, status counts, and ingestion trigger. Recently updated to include new order statuses ("Unpaid/Not Invoiced", "Unpaid/Invoiced") in filters.
    - OrderDetail.jsx: Displays detailed order information, line items, links to BrokerBin, allows manual status updates, and conditionally renders order processors. Recently updated to include buttons for "Send Paid Invoice" and "Send Wire Transfer Invoice" based on order status.
    - DomesticOrderProcessor.jsx: Handles domestic order fulfillment logic.
    - InternationalOrderProcessor.jsx: Handles international order fulfillment logic, including customs details and UPS international shipping.
    - Login.jsx: Handles user login via Firebase.

  - App.jsx: Main application component for routing. Updated with new routes for added features.
  - AuthContext.jsx: Manages Firebase authentication state and provides an apiService for backend calls.
  - UtilitiesLandingPage.jsx: Provides navigation to various utility modules.
  - HpeDescriptionList.jsx & HpeDescriptionForm.jsx / EditHpeDescriptionForm.jsx: For CRUD operations on HPE PO descriptions.
  - CustomsInfoList.jsx & CustomsInfoForm.jsx: New components for CRUD operations on Customs Product Information.
  - SendReceiptForm.jsx: New form for sending "Paid Invoices" (Receipts).
  - SendWireTransferInvoiceForm.jsx: New form for sending "Unpaid (Wire Transfer) Invoices", including a checkbox for an optional $25 fee.
  - Other utility components: StandaloneUpsLabel.jsx, SupplierList.jsx, SupplierForm.jsx, EditSupplierForm.jsx, QuickbooksSync.jsx, DeleteOrderUtil.jsx.
- **Styling:** CSS (e.g., Dashboard.css, OrderDetail.css, FormCommon.css).

Backend:

- **Framework:** Flask (Python)
- **Main file:** app.py (entry point, Flask app initialization, registers blueprints)
- **Blueprints for modular routing:**
  - orders.py: Handles core order processing, ingestion, status updates, and new invoice-sending endpoints.
  - international.py: Handles international order specific details.
  - suppliers.py: CRUD for suppliers.
  - hpe_mappings.py: CRUD for HPE product description mappings.
  - customs_info_crud.py: New blueprint for CRUD operations on the customs_info table.
  - quickbooks.py: For QuickBooks integration.
  - reports.py: For generating reports like daily revenue.

    - utils_routes.py: For miscellaneous utility functions like standalone label generation and order deletion.
- **Key Services:**
    - shipping_service.py: Handles interactions with UPS and FedEx APIs for label generation, shipment processing, and **BigCommerce order status updates**.
    - document_generator.py: Generates PDF documents (POs, Packing Slips, **Paid Invoices, Wire Transfer Invoices**) using ReportLab.
    - email_service.py: Sends emails (POs, notifications, **Paid Invoices, Wire Transfer Invoices**) via Postmark.
    - gcs_service.py: (Mentioned in original report) Manages file uploads/downloads with Google Cloud Storage (for documents like labels, POs, logos).
    - iif_generator.py: (Mentioned in original report) Likely for QuickBooks IIF file generation.

Database:

- **Type:** PostgreSQL (hosted on Google Cloud SQL).
- **ORM/Driver:** SQLAlchemy Core (text-based queries) with pg8000 via Google Cloud SQL Connector.
- **Key Tables:** orders, order_line_items, purchase_orders, po_line_items, shipments, suppliers, hpe_part_mappings, hpe_description_mappings, customs_info (new).

Authentication:

- Firebase Authentication for user login and protecting backend API routes using ID tokens and custom claims (isApproved).

Deployment (Assumed from original report):

- **Backend (Flask):** Google Cloud Run (containerized with Docker).
- **Frontend (React/Vite):** Firebase Hosting or another static site hosting solution.

Cloud Services:

- **Google Cloud Platform (GCP):**
    - Cloud Run (backend hosting).

    - Cloud SQL (PostgreSQL database).
    - Google Cloud Storage (GCS) (storing generated documents, logos).
    - Google Container Registry (GCR) or Artifact Registry (Docker images).
- **Firebase:**
    - Authentication.
    - Potentially Hosting for frontend.

Third-Party APIs:

- BigCommerce API (order ingestion, status updates).
- UPS API (shipping, international documents).
- FedEx API (shipping).
- Postmark API (emails).
- BrokerBin (part lookup links).

3. Key Features Implemented & Steps Taken (Detailed)

This section details features from the original report and integrates new features and modifications from our recent chat session.

- **User Authentication:** Firebase-based login and protected backend routes using ID tokens and an isApproved custom claim check implemented in the @verify_firebase_token decorator in app.py.
- **Order Ingestion (orders.py - ingest_orders_route):**
    - Pulls orders from BigCommerce with a specific processing status ID.
    - Parses customer messages for freight details and compliance IDs.
    - Determines if an order is international.
    - **Recent Update:**
        - Identifies orders with "Bank Wire Transfer" (or similar) payment methods.
        - Sets the initial status for these orders in the local database to "Unpaid/Not Invoiced".

        - Other new orders are set to "new".
        - The finalized_or_manual_statuses list updated to include "Unpaid/Invoiced" to prevent incorrect status reversions.
- **Dashboard (Dashboard.jsx):**
    - Displays orders with filtering by status.
    - Shows status counts fetched from /api/orders/status-counts.
    - Button to trigger /api/ingest_orders.
    - Navigation to Daily Sales report.
    - **Recent Update:**
        - The "Filter by Status" dropdown now includes "Unpaid/Not Invoiced" and "Unpaid/Invoiced".
        - Status badge CSS classes will need corresponding styles for these new statuses (e.g., in Dashboard.css).
        - "Send Receipt" button (previously added here) was **moved** to OrderDetail.jsx.
- **Order Detail View (OrderDetail.jsx):**
    - Displays detailed order information, line items, customer details.
    - Links to BrokerBin for part numbers.
    - Allows manual status updates (e.g., to "RFQ Sent", "Pending").
    - Conditionally renders DomesticOrderProcessor.jsx or InternationalOrderProcessor.jsx.
    - Displays calculated profit for "Processed" orders.
    - **Recent Updates:**
        - **"Send Paid Invoice" Button:** Displayed if order.status is "Processed". Navigates to /orders/:orderId/send-receipt-form.
        - **"Send Wire Transfer Invoice" Button:** Displayed if order.status is "Unpaid/Not Invoiced". Navigates to /orders/:orderId/send-wire-invoice-form.

        - CSS class send-paid-invoice-button added for styling, intended to match back-to-dashboard-button but with a slightly darker blue.

- **Domestic Order Processing (DomesticOrderProcessor.jsx):** Handles single/multi-supplier POs, G1 Onsite fulfillment, generates PO PDFs, Packing Slips, integrates with UPS/FedEx for labels via shipping_service.py.
- **International Order Processing (InternationalOrderProcessor.jsx):** Fetches international details, handles POs, generates UPS international labels and customs forms. Includes logic for InvoiceLineTotal, shipment descriptions, and service code mapping.
- **NEW: Customs Product Information Module:**
    - **Backend (customs_info_crud.py):** New Flask Blueprint created with API endpoints (/api/customs-info, /api/customs-info/<id>) for full CRUD operations on the customs_info table (fields: product_type, customs_description, harmonized_tariff_code, default_country_of_origin, id).
    - **Frontend (CustomsInfoList.jsx, CustomsInfoForm.jsx):** New React components for listing, adding, and editing customs product information, mirroring the structure of the HPE Descriptions module.
    - **Routing (App.jsx):** New routes /admin/customs-info, /admin/customs-info/add, /admin/customs-info/edit/:itemId added.
    - **Utilities Link (UtilitiesLandingPage.jsx):** Link added to navigate to this new module.
- **HPE Product Descriptions Module (hpe_mappings.py, HpeDescriptionList.jsx, HpeDescriptionForm.jsx):** Existing module for CRUD operations on HPE PO descriptions.
- **NEW: Paid Invoice (Receipt) Functionality:**
    - **Trigger:** "Send Paid Invoice" button on OrderDetail.jsx for "Processed" orders.
    - **Frontend Form (SendReceiptForm.jsx):** Collects/confirms recipient email.
    - **Backend API (orders.py - /api/orders/:orderId/send-receipt):**
        - Fetches order details and line item descriptions (including HPE PO descriptions).

        - Calls document_generator.generate_receipt_pdf() (originally named this, then titled "PAID INVOICE", now refined to just "INVOICE" title with a rotated "PAID" stamp).
        - Calls email_service.send_customer_receipt_email() to send the PDF.
    - **PDF Format (document_generator.py - generate_receipt_pdf):**
        - Title: "INVOICE" (top right).
        - Company logo (GCS fetched, 30% larger).
        - Order # (larger, above date), Date, Shipping Method, Payment Method (grouped, right-aligned).
        - Bill To / Ship To addresses (now include customer name).
        - Item Description: Uses hpe_description_mappings.po_description or falls back to item name; SKU/OptionPN/SparePN lines removed from this cell.
        - Rotated red "PAID" text (counter-clockwise 30 degrees, no background box, larger font), centered below addresses.
        - Items table full width.
        - Financial Summary: "TOTAL:" label 25% smaller than its previous larger size; values aligned so labels line up with "Each" column.
        - Removed P.O. Number, Payment Method, Due Date lines from the old info block.
        - Footer: "This invoice has been paid..." message moved up ~1/4 inch, font 15% larger. HR line below it is full width. Fixed company details at bottom.
- **NEW: Unpaid (Wire Transfer) Invoice Functionality:**
    - **Trigger:** "Send Wire Transfer Invoice" button on OrderDetail.jsx for "Unpaid/Not Invoiced" orders.
    - **Frontend Form (SendWireTransferInvoiceForm.jsx):**
        - Displays order summary.
        - Checkbox: "Add $25 Wire Transfer Bank Fee?" (defaults to true).

        - Email input (defaults to customer email).
    - **Backend API (orders.py - /api/orders/:orderId/send-wire-invoice):**
        - Receives email and add_wire_fee flag.
        - Validates order status is "Unpaid/Not Invoiced".
        - Fetches order data, prepares line items (including HPE PO descriptions).
        - Calls document_generator.generate_wire_transfer_invoice_pdf():
            - Passes apply_wire_fee flag.
        - Calls email_service.send_wire_transfer_invoice_email().
        - Updates BigCommerce order status to "Awaiting Payment" (ID 7) via shipping_service.set_bigcommerce_order_status().
        - Updates local G1 PO App order status to "Unpaid/Invoiced".
    - **PDF Format (document_generator.py - generate_wire_transfer_invoice_pdf):**
        - Based on the "Paid Invoice" (now "INVOICE") structure but:
            - No "PAID" stamp.
            - Includes a static section with Wire Transfer bank details (Bank Name, SWIFT, Routing, Account #, Beneficiary, Bank Address) based on Invoice_1305336.pdf.
            - Footer message: "Please send payment using the included payment instructions. Thank you for your order!"
            - If apply_wire_fee is true:
                - Adds a line item: "Bank Wire Transfer Fee" for $25.00.
                - The Grand Total on the PDF includes this $25.00.
            - Header includes "P.O. Number" (from compliance_info), "Terms: Wire Transfer", "Due Date" (invoice date or "Due Upon Receipt").
- **Document Generation (document_generator.py):**

    - Generates POs, Packing Slips, "Paid" Invoices, and "Wire Transfer" Invoices.
    - Supports custom fonts, GCS logo (SVG/PNG/JPG), blind slips.
    - Payment method display on packing slips strips bracketed content.
    - Multiple PDF formatting iterations for "Paid Invoice" and "Wire Transfer Invoice" implemented as detailed above.
- **Shipping Services (shipping_service.py):**
    - Handles OAuth for UPS/FedEx.
    - Generates domestic UPS/FedEx labels.
    - Generates UPS international shipments and documents.
    - Converts GIF labels to PDF.
    - Includes set_bigcommerce_order_status() used by the new wire invoice feature.
    - Canadian province code conversion for UPS international.
- **Email Services (email_service.py):**
    - Sends emails via Postmark.
    - Sends POs, G1 Onsite notifications.
    - send_customer_receipt_email(): For "Paid Invoices".
    - send_wire_transfer_invoice_email(): New function for "Wire Transfer Invoices".
- **Other Backend API Endpoints:** CRUD for Suppliers, HPE Descriptions, Customs Info; Lookups; QuickBooks sync; Daily revenue; Utility routes.

4. Docker & Deployment Workflow

(As per original Handover Report and user interactions)

- Backend is containerized using Docker (Dockerfile in backend root).
- Build: docker build -t gcr.io/<PROJECT_ID>/g1-po-app-backend-service:<TAG>
- Push: docker push gcr.io/<PROJECT_ID>/g1-po-app-backend-service:<TAG>

- Deploy to Cloud Run: gcloud run deploy <SERVICE_NAME> --image gcr.io/<PROJECT_ID>/g1-po-app-backend-service:<TAG> ...

5. Summary of Key Changes & Fixes From This Chat Session

- **Customs Product Information Module:** Fully scoped and code structure provided for backend (Flask blueprint, API endpoints for CRUD on customs_info table) and frontend (React components CustomsInfoList.jsx, CustomsInfoForm.jsx, routing in App.jsx, link in UtilitiesLandingPage.jsx).
- **Paid Invoice (Receipt) Feature:**
    - Button added to OrderDetail.jsx for "Processed" orders.
    - SendReceiptForm.jsx created for email input.
    - Backend API endpoint /api/orders/:orderId/send-receipt created in orders.py.
    - document_generator.py updated with generate_receipt_pdf (iteratively refined format: title "INVOICE", GCS logo, address/contact details, item descriptions from HPE mappings, rotated "PAID" stamp, full-width items table, specific summary alignment, fixed footers).
    - email_service.py updated with send_customer_receipt_email.
- **Unpaid (Wire Transfer) Invoice Feature:**
    - **New Statuses:** "Unpaid/Not Invoiced", "Unpaid/Invoiced" defined.
    - **Order Ingestion (orders.py):** Modified ingest_orders_route to identify wire transfer payment methods and set initial order status to "Unpaid/Not Invoiced".
    - **Dashboard (Dashboard.jsx):** Updated orderedDropdownStatuses to include new statuses for filtering. get_order_status_counts in orders.py also updated.
    - **Order Detail (OrderDetail.jsx):** "Send Wire Transfer Invoice" button added for "Unpaid/Not Invoiced" orders.
    - **New Form (SendWireTransferInvoiceForm.jsx):** Created with recipient email input and a checkbox (defaulting to true) for adding a $25 wire transfer fee.
    - **New Backend API (orders.py - /api/orders/:orderId/send-wire-invoice):**

        - Accepts email and add_wire_fee flag.
        - Calls new document_generator.generate_wire_transfer_invoice_pdf().
        - Calls new email_service.send_wire_transfer_invoice_email().
        - Updates BigCommerce order status to "Awaiting Payment" (ID 7) via shipping_service.set_bigcommerce_order_status().
        - Updates local order status to "Unpaid/Invoiced".
    - **New PDF Function (document_generator.py - generate_wire_transfer_invoice_pdf):**
        - Title: "INVOICE".
        - No "PAID" stamp.
        - Includes static wire transfer bank details section.
        - Footer message: "Please send payment using the included payment instructions..."
        - Conditionally adds $25 "Bank Wire Transfer Fee" line item and adjusts total if apply_wire_fee is true.
        - Includes PO Number, "Terms: Wire Transfer", and Due Date in header details.
        - Incorporates other recent formatting requests (logo size, Invoice # size/position, summary alignment, footer position).
- **Troubleshooting & Code Corrections:**
    - document_generator.py: Added from decimal import Decimal. Corrected COMPANY_LOGO_GCS_URI usage in if __name__ == '__main__' test block. Fixed AttributeError for Label.getText() to Label.text and then to using shapes.String for rotated text. Fixed KeyError for missing ParagraphStyle definitions by adding them to get_custom_styles().
    - orders.py: Corrected SQL parameter style from %s to :param_name for SQLAlchemy text() queries. Fixed "column original_sku does not exist" by aliasing oli.sku AS original_sku. Added missing imports (os, date).
    - app.py: Fixed Firebase Admin SDK double initialization error by checking if not firebase_admin._DEFAULT_APP_NAME in firebase_admin._apps:.

- App.jsx: Ensured SendWireTransferInvoiceForm component was correctly imported after being identified as "not defined".
- **GCS Authentication:** Advised user to run gcloud auth application-default login to fix "Reauthentication is needed" errors when fetching logo from GCS.
- **.env Loading:** Advised adding load_dotenv() to document_generator.py for local testing if COMPANY_LOGO_GCS_URI wasn't being picked up from .env.
- **curl Command:** Provided correct syntax for Windows cmd.exe (escaping quotes in JSON data) and explained difference between Firebase Web API Key and ID Token.
- **Browser Console Errors (Frontend):**
  - Addressed Firebase auth/quota-exceeded and 400 Bad Request to securetoken.googleapis.com by advising verification of Firebase frontend config API key, GCP API key restrictions, and Identity Toolkit API enablement.
  - Addressed validateDOMNesting warning in Dashboard.jsx by advising removal of whitespace text nodes from within <tr> tags.
- **Clarified Misunderstandings:**
  - Relocated "Send Receipt" button from Dashboard.jsx to OrderDetail.jsx.
  - Confirmed (then re-confirmed) logic for the $25 wire transfer fee (user-selectable checkbox is needed as BC does not add it to total).

6. Remaining Tasks & Future Considerations (Detailed Plan)

This section builds upon the original handover report's "Remaining Tasks" and incorporates new items.

A. Immediate Next Steps (Post-Chat Session):

1. **Finalize Frontend for Wire Transfer Invoice:**
   - **OrderDetail.jsx:** Ensure the "Send Wire Transfer Invoice" button is correctly implemented and styled as per latest requests (button style matching, placement).

- **SendWireTransferInvoiceForm.jsx:** Ensure this form is complete, functional, correctly passes add_wire_fee to the backend, and handles UI updates/navigation.
- **App.jsx Routing:** Confirm the route /orders/:orderId/send-wire-invoice-form is correctly defined and uses ProtectedRoute.
- **Dashboard.jsx Statuses:** Verify that the new statuses ("Unpaid/Not Invoiced", "Unpaid/Invoiced") display correctly in the table and that the "Unpaid/Invoiced" orders can be filtered or grouped with "New" orders for processing workflow.

2. **Thorough End-to-End Testing of New Invoice Features:**
   - **Wire Transfer Invoice Flow:**
     - Ingestion of a wire transfer order from BC -> Sets status to "Unpaid/Not Invoiced".
     - Navigation from OrderDetail.jsx to SendWireTransferInvoiceForm.jsx.
     - Form submission with and without the $25 fee.
     - Verify PDF content and total calculation for both fee scenarios.
     - Verify email content and delivery.
     - Verify BigCommerce status updates to "Awaiting Payment" (ID 7).
     - Verify local G1 PO App status updates to "Unpaid/Invoiced".
     - Confirm "Unpaid/Invoiced" orders are actionable for fulfillment (like "New" orders).
   - **Paid Invoice (Receipt) Flow:**
     - Ensure the "Send Paid Invoice" button on OrderDetail.jsx (for "Processed" orders) works.
     - Verify SendReceiptForm.jsx functionality.
     - Verify PDF and email for paid invoices.
3. **BigCommerce Status Update Verification:** The last point of failure was the BigCommerce status update not occurring for the wire transfer invoice. After ensuring shipping_service.py has detailed logging (as discussed), re-test this specific step. If it still fails, examine the logs from

shipping_service.set_bigcommerce_order_status to see the exact response from BigCommerce. This might involve checking API permissions or if Status ID 7 is valid for the state transition.

B. From Original Handover Report & New Considerations:

- **Thorough Testing of All Existing Scenarios:**
    - Domestic: Single Supplier, Multi-Supplier, G1 Onsite (Bill Sender, Bill Recipient UPS/FedEx, Bill Third Party UPS).
    - International: Supplier PO, G1 Onsite (Bill Sender, Bill Receiver UPS - especially with postal code for BillReceiver.Address).
    - All document generation (POs, Packing Slips for various scenarios including blind).
    - All email notifications.
    - Edge cases for InvoiceLineTotal (e.g., $0 value shipments to CA/PR).
    - Various shipping method mappings from BigCommerce.
- **FedEx International Shipments:**
    - Currently, InternationalOrderProcessor.jsx is UPS-centric.
    - If FedEx international is required, build out payload construction, service code mapping, and billing type handling in InternationalOrderProcessor.jsx and create a corresponding generate_fedex_international_shipment function in shipping_service.py.
- **"Bill Third Party" for UPS International:**
    - Clarify if a true third-party (not shipper or receiver) billing scenario is needed.
    - If so, UI in InternationalOrderProcessor.jsx needs to collect distinct third-party address/postal code. The current implementation uses receiver details for BillReceiver.
- **UI/UX Refinements:**
    - Ensure consistent error display and user feedback across all new forms and actions.
    - Implement and verify loading indicators for all new async operations.

    - Potentially enhance OrderDetail.jsx to show links to sent invoices (Paid or Wire Transfer) if their GCS paths are stored.
- **Refined Profitability Calculation (International):** The current calculation is basic. Needs to incorporate actual costs if available.
- **Configuration Management:** Review and document all environment variables, especially new ones, for production.
- **Code Cleanup & Optimization:**
    - Refactor shared frontend helper functions into utils.js.
    - Review Python backend for redundancy.
- **Comprehensive Logging & Monitoring (Production):**
    - Transition from print() to a structured logging library (e.g., Python's logging module configured in Flask) for better production log management.
    - Set up monitoring and alerting.
- **Database Schema Review:**
    - Consider if orders table needs a field to store the invoiced amount if the $25 wire fee is applied (since total_sale_price reflects the original BC total). This could be useful for reconciliation.
    - Consider fields in shipments or a new table to track sent invoices (type, date, recipient, PDF GCS path).
- **Security Hardening:** Review all new API endpoints, authentication/authorization, and input validation.
- **QuickBooks Integration (iif_generator.py):** Further development and testing if this is a core ongoing requirement.

7. Technical Details for Developer

(Mostly from original Handover Report, updated for clarity)

- **Backend Entry Point:** app.py.
- **Frontend Entry Point:** src/main.jsx.
- **Key Data Flow for New Invoice Features:**

1. **Ingestion:** Dashboard.jsx -> POST /api/ingest_orders (orders.py) -> BigCommerce API -> orders table (sets status to "Unpaid/Not Invoiced" for wire transfers).
2. **User Navigates to Order Detail:** Dashboard.jsx -> OrderDetail.jsx (/orders/:orderId).
3. **User Initiates Invoice Send (Wire Transfer Example):**
    - OrderDetail.jsx (if status "Unpaid/Not Invoiced") -> "Send Wire Transfer Invoice" button -> Navigates to SendWireTransferInvoiceForm.jsx (/orders/:orderId/send-wire-invoice-form).
4. **Invoice Form Submission:**
    - SendWireTransferInvoiceForm.jsx -> POST /api/orders/:orderId/send-wire-invoice (orders.py) with email and add_wire_fee flag.
5. **Backend send_wire_invoice_route (orders.py):**
    - Fetches order data.
    - Calls document_generator.generate_wire_transfer_invoice_pdf().
    - Calls email_service.send_wire_transfer_invoice_email().
    - Calls shipping_service.set_bigcommerce_order_status() (to Status ID 7).
    - Updates local order status to "Unpaid/Invoiced".
6. (Similar flow for "Paid Invoice" / Receipt, starting from "Processed" orders and using send_order_receipt_route).

- **Environment Variables:** Critical. .env for local. Ensure all (DB, Firebase, GCS, BigCommerce, Postmark, Shipping APIs, new status IDs if configurable) are set for deployment.
- **API Versions:** UPS API v2409. FedEx versions should be noted if specific. BigCommerce is v2 for orders.
- **State Management (Frontend):** Component-level useState/useEffect; AuthContext for global user/API service.

- **Backend Database Interactions:** SQLAlchemy Core with text(). Transactions for atomic operations.

This expanded report should provide the next developer with a clear understanding of the project's current state, the recent feature additions, and the path forward.