G1 PO App - Definitive Handover Report & Project Plan (v12.0)

Date: May 18, 2025 (Reflecting all progress and chat sessions ending May 18, 2025)
Version: 12.0 (This document supersedes v11.0 and incorporates all recent development, troubleshooting, and planning, including FedEx integration groundwork and QuickBooks integration 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, recent progress, known issues, and a detailed plan for future development and completion. It is intended to facilitate a smooth handover to an incoming developer.

Table of Contents

    - **Email Service (email_service.py)**
    - **Document Generator (document_generator.py)**
    - **IIF Generator (iif_generator.py)**
    - **Frontend (App.jsx, AuthContext.jsx, Dashboard.jsx, OrderDetail.jsx, UtilitiesLandingPage.jsx, HpeDescriptionList.jsx, HpeDescriptionForm.jsx, EditHpeDescriptionForm.jsx, StandaloneUpsLabel.jsx, QuickbooksSync.jsx (placeholder))**
5. **Database Schema Notes & Key Tables (Highlighting Recent Changes)**
    - orders Table (UPS & FedEx customer billing fields, QuickBooks sync status)
    - shipments Table
    - purchase_orders Table (QuickBooks sync status)
    - hpe_description_mappings Table (Now actively managed)
    - Other relevant tables
6. **Developer Setup & Environment**
    - Backend
    - Frontend
    - Running Locally
    - Deployment (Cloud Run for Backend, Firebase Hosting for Frontend)
    - Key Environment Variables
7. **Detailed Plan for Completion & Future Tasks (Updated & Prioritized)**
    - **Phase 1: Immediate Post-Handover Tasks & Verification**
        - FedEx "Bill Recipient" Sandbox Testing (Critical)
        - FedEx Production Label Certification (Critical)
        - GCS Signed URL Generation (Ongoing Troubleshooting from v11.0)
        - Comprehensive E2E Testing of All Fulfillment Flows (UPS, FedEx, G1 Onsite)

- **Phase 2: QuickBooks Integration Module (New Major Feature)**
  - Task 1: Finalize QuickBooks Version & Integration Strategy
  - Task 2: Database Schema for QuickBooks Sync Status
  - Task 3: Backend Logic for Sales Order Data Extraction (from orders table)
  - Task 4: Backend Logic for Purchase Order Data Extraction (from purchase_orders table)
  - Task 5: IIF File Generation (or API client if QuickBooks Online) for Sales Orders
  - Task 6: Modify IIF File Generation for Purchase Orders (if needed, or reuse)
  - Task 7: Backend API Endpoint for On-Demand QuickBooks Sync
  - Task 8: Frontend UI for QuickBooks Sync (QuickbooksSync.jsx)
  - Task 9: Thorough Testing of QuickBooks Sync
- **Phase 3: Finalizing FedEx Integration (Post-Certification)**
  - Production "Bill Recipient" Testing with Real Customer Accounts (Pilot)
  - Integrate FedEx Label Generation into app.py/process_order
  - (Optional) Standalone FedEx Label Generator UI
- **Phase 4: UI/UX Enhancements & Remaining Fixes (from original report, re-prioritized)**
  - Supplier Management UI (CRUD - SupplierList.jsx, SupplierForm.jsx, EditSupplierForm.jsx) - *Partially complete via Utilities page structure*
  - HPE Description Management UI (CRUD - HpeDescriptionList.jsx, etc.) - *Good progress made*
  - Frontend - Eloquia Font Issue (from v11.0)
  - Loading Indicators and UX Refinements (General)
- **Phase 5: MVP Task Completion (from original report, re-prioritized)**

(Content for each section will now be detailed, incorporating previous report info and our chat progress)

(The content below would be a merge and update of your v11.0 document and our detailed chat. Due to length constraints here, I will focus on summarizing the new additions and the detailed QuickBooks plan, assuming the foundational details from v11.0 are available to the incoming developer from that document.)

1. Project Goal & Core Value Proposition

- **(As per v11.0, with additions):** 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.
- **Key Functionality:** Ingests orders from BigCommerce, allows authorized users to manage orders, generate POs, Packing Slips, Shipping Labels (UPS & **FedEx**), communicate with suppliers, update BigCommerce, and integrate with QuickBooks Desktop for accounting (POs and **Sales Orders**).
- **Recent Enhancements Include (from v11.0 and this chat):**
    - "G1 Onsite Fulfillment."
    - "Customer-Billed UPS Shipments."
    - "Standalone UPS Label Generator."
    - **Groundwork for "Customer-Billed FedEx Shipments."**
    - **Planned: Enhanced QuickBooks integration for Sales Orders and on-demand sync.**
    - **UI Reorganization:** Introduction of a "Utilities" section for administrative tools.
- **Primary Aim:** Reduce manual effort, improve accuracy, provide a centralized platform for critical business workflows, and replace functionalities of external tools like T-HUB for BigCommerce-to-QuickBooks order sync.

2. Current State of the Application (As of May 18, 2025)

- **(Summary from v11.0 Key Achievements, updated with this chat's progress):**
    - **Order Ingestion & Management:** Successfully ingests BigCommerce orders, parses customer notes for UPS and **FedEx** third-party billing details. Dashboard for viewing/filtering.
    - **Dashboard UI:** Consolidated "Orders" and "Daily Sales Report."
    - **Order Detail UI:** Displays detailed order info, supports single/multi-supplier POs, G1 Onsite Fulfillment, and shows UPS/FedEx customer account info.
    - **Backend Processing (app.py):** Handles order processing, multi-supplier logic, G1 Onsite, and passes data for customer-billed UPS/FedEx. Generates unique PO numbers.

- **Document Generation:** PDFs for POs and Packing Slips.
- **Shipping Label Generation:**
  - **UPS (shipping_service.py):** Fully functional for "Bill Sender" and "Bill Third Party" (customer account) using OAuth 2.0. Generates PDF labels. Sandbox tested.
  - **FedEx (shipping_service.py):**
    - OAuth 2.0 token retrieval for **sandbox AND production** is functional (scope CXS-TP achieved).
    - Service code mapping implemented.
    - generate_fedex_label_raw and generate_fedex_label drafted.
    - **"Bill SENDER" successfully tested against FedEx PRODUCTION, generating a live (test-marked) label.**
    - "Bill RECIPIENT" sandbox testing for accounts *other than self* results in ACCOUNT.VALIDATION.FAILED, awaiting FedEx guidance/test accounts. Production "Bill RECIPIENT" not yet live tested.
- **Email Communication (Postmark):** Sends POs, Packing Slips, UPS Labels. send_sales_notification_email for G1 Onsite and Standalone UPS labels.
- **Cloud Storage (GCS):** Uploads generated documents. Signed URL generation troubleshooting was ongoing (see v11.0).
- **BigCommerce Integration:** Retrieves order data, creates shipments, updates order statuses.
- **QuickBooks Integration (Current - IIF for POs):** Generates and emails daily IIF files for *Purchase Orders* (yesterday's via scheduler, today's via user trigger).
- **Authentication & Authorization (Firebase):** Robust Google Sign-In, Firebase Custom Claims (isApproved), backend API protection. AuthContext.jsx provides apiService for authenticated frontend calls.
- **Frontend UI Structure:**

        - Navigation updated to "Orders | Daily Sales | Utilities".
        - UtilitiesLandingPage.jsx created to house links to:
            - Standalone UPS Label Generator (functional).
            - QuickBooks Integration (placeholder component QuickbooksSync.jsx created, route defined).
            - Supplier Management (links to SupplierList.jsx - API exists).
            - **HPE Description Management** (repurposed from Product Management, links to HpeDescriptionList.jsx - API and basic frontend CRUD functional after significant refactoring in this session, including pagination and Option PN filtering).
- **Recent Developments & Enhancements (Post v11.0 Report - from this chat session):**
    - **FedEx Integration Groundwork:**
        - Database schema updated for FedEx customer billing fields in orders table.
        - app.py (ingest_orders) updated to parse FedEx account details from BigCommerce comments.
        - shipping_service.py:
            - Added FedEx production and sandbox credential/URL configuration.
            - Implemented get_fedex_oauth_token() (now working for CXS-TP scope).
            - Implemented map_shipping_method_to_fedex_code() using official ENUMs.
            - Drafted generate_fedex_label_raw() and generate_fedex_label() .
            - **Successfully generated a "Bill SENDER" label against FedEx Production API (though label was test-marked).**
            - Diagnosed "Bill RECIPIENT" sandbox failures to be account validation/permission issues, not code structure.

- **CORS Issues Resolution:**
  - Addressed multiple CORS preflight (OPTIONS) request failures by:
    - Correcting frontend URL construction to avoid double /api/api/.
    - Modifying verify_firebase_token decorator in app.py to correctly handle OPTIONS requests using current_app.make_default_options_response().
    - Ensuring relevant routes in app.py list OPTIONS in their methods array.
- **HPE Description Management Refactor:**
  - Backend /api/products routes in app.py were successfully refactored to /api/hpe-descriptions and now manage the hpe_description_mappings table (CRUD operations for option_pn, po_description).
  - Frontend components (ProductMappingList.jsx, ProductMappingForm.jsx, EditProductMappingForm.jsx) were conceptually refactored and renamed to HpeDescriptionList.jsx, HpeDescriptionForm.jsx, EditHpeDescriptionForm.jsx to align with the backend changes.
  - Pagination and server-side filtering by OptionPN implemented for HpeDescriptionList.jsx and its backend endpoint.
- **Frontend UI Reorganization:**
  - Implemented the "Utilities" section in the main navigation.
  - Created UtilitiesLandingPage.jsx with cards linking to existing and planned utilities.
  - Corrected import paths and component calls related to apiService from AuthContext.jsx (e.g., using apiService.get(), apiService.post() instead of calling apiService() directly).

(Sections 3, 4, 5, 6 would be largely similar to your v11.0 report, but with updates for new FedEx config variables, new frontend components, database changes for FedEx and QB sync, and any new libraries if introduced. I will highlight key additions here.)

3. Technology Stack & Project Layout
* **External APIs (Additions):** FedEx Ship API (OAuth 2.0).
* **Frontend Build:** Vite continues to be used.

4. Key Code Files & Their Roles (Highlighting Recent Changes & Additions)
* **app.py:**
* Modified ingest_orders for FedEx details.
* /api/products/* routes refactored to /api/hpe-descriptions/* for
managing hpe_description_mappings, including pagination and filtering for the list view.
* verify_firebase_token decorator updated for robust OPTIONS handling for CORS.
* **NEW (To be added):** /api/quickbooks/trigger-sync endpoint.
* **shipping_service.py:**
* Added FedEx production/sandbox config.
* Added get_fedex_oauth_token(), map_shipping_method_to_fedex_code().
* Added generate_fedex_label_raw() and generate_fedex_label() (Bill SENDER works in prod, Bill RECIPIENT structure is there).
* **iif_generator.py:** Currently handles POs. Will need review/extension for Sales Orders if IIF is chosen for them.
* **Frontend Components:**
* App.jsx: Main navigation updated. Routes for new Utilities section added.
* AuthContext.jsx: apiService object provides .get(), .post(), .put(), .delete() methods.
* **NEW:** UtilitiesLandingPage.jsx (acts as a dashboard for utility modules).
* HpeDescriptionList.jsx, HpeDescriptionForm.jsx, EditHpeDescriptionForm.jsx (refactore
d from product mapping components, now functional for hpe_description_mappings with pagination/filtering).
* StandaloneUpsLabel.jsx (existing).
* **NEW (Placeholder):** QuickbooksSync.jsx (created, basic placeholder, route defined).

5. Database Schema Notes & Key Tables
* orders Table:
* Added customer_fedex_account_number (VARCHAR)
* Added is_bill_to_customer_fedex_account (BOOLEAN)
* Added customer_selected_fedex_service (VARCHAR)
* Added customer_fedex_account_zipcode (VARCHAR, currently NULL)
* **NEW (To be added):** qb_sales_order_sync_status (e.g., VARCHAR: 'pending', 'synced', 'error',
'skipped'), qb_sales_order_synced_at (TIMESTAMP), qb_sales_order_last_error (TEXT).
* purchase_orders Table:
* **NEW (To be**

added): qb_po_sync_status (VARCHAR), qb_po_synced_at (TIMESTAMP), qb_po_last_error (TEXT).
* hpe_description_mappings Table: Now actively managed by the UI.

6. Developer Setup & Environment
* **New Environment Variables (Backend .env / Secret Manager):**
* FEDEX_API_KEY_SANDBOX, FEDEX_SECRET_KEY_SANDBOX, FEDEX_ACCOUNT_NUMBER_SANDBOX
* FEDEX_API_KEY_PRODUCTION, FEDEX_SECRET_KEY_PRODUCTION, FEDEX_ACCOUNT_NUMBER_PRODUCTION
* FEDEX_API_ENVIRONMENT (e.g., "sandbox" or "production")
* FEDEX_GRANT_TYPE (e.g., "client_credentials")
* FEDEX_OAUTH_URL_SANDBOX, FEDEX_OAUTH_URL_PRODUCTION
* FEDEX_SHIP_API_URL_SANDBOX, FEDEX_SHIP_API_URL_PRODUCTION
* **Frontend (.env.development, .env.production):**
* VITE_API_BASE_URL must be the root URL of the backend (e.g., https://your-cloud-run-service.run.app, NOT ending in /api).

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

*   **Phase 1: Immediate Post-Handover Tasks & Verification**

   *   **FedEx "Bill Recipient" Sandbox Testing (Critical):**

      *   Action: Engage FedEx Developer Support to obtain valid sandbox recipient account numbers or procedures for testing third-party billing where the recipient account is different from the shipper account.

      *   Goal: Successfully generate a sandbox FedEx label using `paymentType: RECIPIENT` with a true third-party test account.

      *   Files: `shipping_service.py` (testing block).

   *   **FedEx Production Label Certification (Critical):**

      *   Action: Follow FedEx's "Shipping Label Certification steps." Generate sample sandbox labels (PDF/ZPL as appropriate for intended printers) for all services to be used in production. Fill out and submit the Label Cover Sheet with PRODUCTION API Key and Account Number.

      *   Goal: Receive approval from FedEx Label Analysis Group, which authorizes production credentials for live shipping.

        *   Files: `shipping_service.py` (for generating sample labels from sandbox).

    *   **GCS Signed URL Generation (Ongoing Troubleshooting from v11.0):**

        *   Action: Resolve the "private key needed" error for GCS signed URLs. Investigate Cloud Run service account identity, IAM propagation, and direct `Blob.generate_signed_url()` vs. `storage.
प्रतिशतBucket(bucket_name).generate_signed_url()` if using default creds impersonation.

        *   Files: `app.py` (where signed URLs are generated), GCP IAM.

    *   **Comprehensive E2E Testing of All Fulfillment Flows:**

        *   Action: Once FedEx is certified, thoroughly test: G1 Onsite (UPS), Supplier PO (UPS), Customer-Billed UPS, Customer-Billed FedEx (initially as Bill Sender in Prod, then true Bill Recipient post-pilot).

        *   Goal: Ensure all documents are generated correctly, emails sent, BigCommerce updated, and labels are valid.

        *   Files: Full application stack.

*   **Phase 2: QuickBooks Integration Module (New Major Feature)**

    *   **Overall Goal:** Replicate T-HUB's BigCommerce-to-QuickBooks sales order sync. Add on-demand sync for POs and Sales Orders. Track sync status.

    *   **Assumed:** QuickBooks Desktop, primary integration via IIF files (unless a decision is made to explore QBWC or QB Online API).

    *   **Task 1: Finalize QuickBooks Version & Integration Strategy Detail**

        *   Action: Confirm definitively if QuickBooks Desktop or Online is the target. If Desktop, re-evaluate if IIF is sufficient for sales orders vs. the complexity/benefit of QBWC. For this plan, we'll assume IIF for Desktop.

        *   Goal: Clear decision on the integration method.

    *   **Task 2: Database Schema for QuickBooks Sync Status**

    *   Action: Add new columns to `orders` table: `qb_sales_order_sync_status` (VARCHAR: 'pending_sync', 'synced', 'error', 'skipped'), `qb_sales_order_synced_at` (TIMESTAMP), `qb_sales_order_last_error` (TEXT).

    *   Action: Add similar columns to `purchase_orders` table: `qb_po_sync_status`, `qb_po_synced_at`, `qb_po_last_error`.

    *   Default `qb_..._sync_status` to `'pending_sync'` for new relevant records.

    *   Files: Database migration scripts.

*   **Task 3: Backend Logic for Sales Order Data Extraction (from `orders` table)**

    *   Action: Create a function in `app.py` or a new `quickbooks_service.py` to fetch `orders` records that are eligible for QuickBooks sync (e.g., based on their main `status` like 'Completed Offline' or 'Shipped', AND where `qb_sales_order_sync_status` is 'pending_sync' or 'error').

    *   Action: Map data from the `orders` and `order_line_items` tables to the fields required for a QuickBooks Sales Receipt or Invoice (Customer, Item, Qty, Rate, Amount, Tax, Shipping, Payment details if applicable, SO Number, Date). This requires understanding the current T-HUB mapping.

    *   Goal: Structured data ready for IIF generation.

    *   Files: `app.py` or `quickbooks_service.py`.

*   **Task 4: Backend Logic for Purchase Order Data Extraction (from `purchase_orders` table)**

    *   Action: Similar to Task 3, but for `purchase_orders` and `po_line_items`. Fetch POs where `qb_po_sync_status` is 'pending_sync' or 'error'.

    *   Goal: Structured PO data ready for IIF generation.

    *   Files: `app.py` or `quickbooks_service.py`.

*   **Task 5: IIF File Generation for Sales Orders**

    *   Action: Extend `iif_generator.py` (or create a new function) to generate IIF transactions for Sales Receipts or Invoices based on the extracted sales order data. This

will involve understanding IIF spec for these transaction types (e.g., `TRNS`, `SPL`, `ENDTRNS` lines for sales).

 * Action: Handle customer mapping (create new if not existing in QB based on IIF import behavior) and item mapping (ensure item names/SKUs match QB items).

 * Goal: Valid IIF file content for sales orders.

 * Files: `iif_generator.py`.

* **Task 6: Modify IIF File Generation for Purchase Orders (if needed)**

 * Action: Review existing PO IIF generation. Ensure it aligns with the new on-demand sync model and status tracking. It might be reusable as is, just triggered differently.

 * Goal: Consistent IIF generation for POs.

 * Files: `iif_generator.py`.

* **Task 7: Backend API Endpoint for On-Demand QuickBooks Sync**

 * Action: Create a new route in `app.py`, e.g., `POST /api/quickbooks/trigger-sync`.

 * This endpoint will:

  * Call the data extraction functions (Task 3 & 4).

  * Call the IIF generation functions (Task 5 & 6).

  * Combine IIF content if desired, or prepare separate files.

  * Email the IIF file(s) to the designated accounting email.

  * **Crucially:** After successful IIF generation/email, update the `qb_..._sync_status` to 'synced' and `qb_..._synced_at` for the processed records in the database. If errors occur during generation, log them in `qb_..._last_error` and set status to 'error'.

 * Goal: A backend trigger for the complete sync process.

 * Files: `app.py`, `iif_generator.py`, `email_service.py`.

* **Task 8: Frontend UI for QuickBooks Sync (`QuickbooksSync.jsx`)**

        *   Action: Develop the UI in `QuickbooksSync.jsx` (linked from `UtilitiesLandingPage.jsx`).

        *   UI should include:

            *   A "Sync Now to QuickBooks" button.

            *   Display area for last sync status, number of records synced, any errors.

            *   Optional: Date range selectors for syncing specific periods (more advanced).

            *   Optional: Display counts of pending POs/Sales Orders.

        *   The button will call the `/api/quickbooks/trigger-sync` backend endpoint.

        *   Goal: User-friendly interface to initiate and monitor QuickBooks sync.

        *   Files: `QuickbooksSync.jsx`, `App.css`.

    *   **Task 9: Thorough Testing of QuickBooks Sync**

        *   Action: Test with various scenarios: new customers, existing customers, different product types, orders with/without tax and shipping.

        *   Action: Verify IIF file import into a TEST QuickBooks Desktop company file. Check for data accuracy, correct transaction types, customer creation, item linking.

        *   Action: Test error handling and status updates in the database and UI.

        *   Goal: Robust and accurate QuickBooks data transfer.

*   **(Phases 3-7 would then follow, largely based on v11.0 plan but re-prioritized and incorporating the outcomes of Phase 1 & 2. I will summarize here, assuming details are in v11.0 or can be elaborated by the new developer.)**

*   **Phase 3: Finalizing FedEx Integration (Post-Certification)**

    *   Integrate successful "Bill Recipient" (FedEx) into `app.py`/`process_order`.

    *   Build Standalone FedEx Label UI if desired.

*   **Phase 4: UI/UX Enhancements & Remaining Fixes**
    *   Complete Supplier CRUD UI.
    *   Complete HPE Description CRUD UI (List view is good, forms need final polish).
    *   Address Eloquia Font issue.
    *   General UX review for loading states, error messages.

*   **Phase 5: MVP Task Completion**
    *   Implement PO Export (Excel).
    *   Final comprehensive review of AuthN/AuthZ.

*   **Phase 6: Infrastructure & Deployment Finalization**
    *   Ensure Postmark DKIM/SPF is fully set up for production domain.
    *   Configure Cloudflare DNS if using custom domains for frontend/backend.

*   **Phase 7: Long-Term Improvements & Best Practices**
    *   Address Daily Revenue Timezone.
    *   Security Hardening (IAP, input validation, structured logging).
    *   Consider Google Cloud Tasks for asynchronous `process_order` calls.
    *   Build Admin UI for Firebase Custom Claims management.
    *   Ensure QuickBooks List data integrity (TERMS, SHIPVIA for IIFs).
    *   Implement automated tests (Pytest, Jest/React Testing Library).

8. Known Issues & Considerations (Updated from v11.0 and this chat)

- **FedEx "Bill Recipient" Sandbox:** Requires FedEx Support to provide usable test recipient accounts or clarify sandbox setup for third-party billing with account 740561073. (Code structure is ready).

- **FedEx Production Certification:** Label evaluation process is pending. Production credentials will not be fully active for shipping until this is complete.
- **GCS Signed URL Generation:** (Issue from v11.0) Persisting "private key needed" error. Needs investigation of Cloud Run SA identity, IAM, and the method of URL generation.
- **QuickBooks Desktop & IIF Limitations:** IIF is a fragile format. It doesn't provide good feedback on import errors, cannot easily update existing QB transactions, and handling inventory/payments can be complex. If issues arise, QBWC or QB Online API (if client migrates) are more robust alternatives.
- **Scalability of Synchronous Processing:** Current process_order is synchronous. For higher volumes, consider moving label/document generation and external API calls to asynchronous tasks (Phase 7).
- **Order Comment Parsing Reliability:** Parsing freight details from customer_message relies on a specific string format. Consider BigCommerce Order Metafields for a more robust solution if comment parsing becomes error-prone.
- **(Other items from v11.0's Known Issues list remain relevant unless explicitly resolved).**

9. Recent Bug Fixes & Specific File Updates (Summarized)

- **(From v11.0 - still relevant)**
- **This Chat Session:**
    - Corrected FedEx OAuth token acquisition (CXS-TP scope achieved using correct endpoint).
    - Diagnosed and provided path to fix FedEx "Bill Recipient" sandbox account validation.
    - Refactored /api/products to /api/hpe-descriptions in app.py for hpe_description_mappings table, including pagination and filtering.
    - Refactored frontend ProductMappingList/Form components to HpeDescriptionList/Form to align.
    - Resolved multiple CORS preflight (OPTIONS) errors by:

        - Correcting VITE_API_BASE_URL and frontend apiPath concatenation.
        - Updating verify_firebase_token decorator in app.py to correctly handle OPTIONS requests.
    - Corrected frontend calls to apiService (e.g., apiService.get() instead of apiService()).

10. Conclusion for Handover

The G1 PO App has undergone significant development and is functional for several core workflows, including UPS label generation and basic Purchase Order IIF export. Groundwork for FedEx integration is well underway, with production "Bill SENDER" label generation achieved in sandbox mode (test-marked label). The UI has been reorganized with a "Utilities" section, and management for HPE PO Descriptions is now functional with pagination and filtering.

Immediate priorities for the incoming developer are to work with FedEx to complete the production label certification and resolve sandbox testing for "Bill Recipient" with true third-party accounts. Following that, the major new feature is the QuickBooks Integration module, focusing on syncing BigCommerce Sales Orders and providing an on-demand sync mechanism.

The backend API is reasonably well-structured, and the frontend uses React with Firebase for authentication. This document, along with the codebase and previous handover reports, should provide a solid basis for continuing development. Key areas for future improvement include more robust error handling, asynchronous processing for long-running tasks, and comprehensive automated testing.

This report is designed to be as thorough as possible. The incoming developer should review this alongside the v11.0 report and the codebase itself. Good luck to them!