Verizon Project Management Tool: System Design & Implementation Plan

1.0 Executive Summary

This document outlines the system design for a secure, local-first Project Management (PM) Tool for Verizon. The current spreadsheet-based tracking system is inefficient and does not scale for project managers handling 500+ projects, nor does it provide the necessary KPI reporting for management.

This new tool will solve these problems by providing a high-performance desktop application that runs 100% within the Verizon network. It requires no new database servers or cloud APIs. It will use a local SQLite database for speed and a secure "Inbox" synchronization model to safely consolidate data to a master database on the shared G: drive, preventing all data corruption.

The system will provide role-based dashboards, automated KPI reporting, and an on-device AI assistant. This will dramatically reduce administrative overhead, provide instant and accurate reports for all management levels (from Sr. PM to Director), and introduce AI-driven insights—all while adhering to the strictest security constraints.

2.0 Core Architecture (The "Safe Way")

The application is a **standalone desktop application** (.exe) built in Python. This model is 100% secure and requires no server installation.

- **Primary Storage (Local):** Each user runs the app on their local machine. Their *working database* (my_projects.db) is stored on their local **C: drive**. This is the most critical design choice. It guarantees the app is fast and **eliminates all file-locking and "database is locked" errors** that corrupt data on network drives.
- **Shared Storage (Network):** The shared **G: drive** is used *only* as a secure "post office" for data synchronization and to hold central configuration files. The app *never* runs a live database from the G: drive.

3.0 Security & Access Control

Access is controlled by a mandatory **login system**. This allows for a single application to securely serve all roles.

- **User Database:** A central master_users.db on the G: drive will store user credentials (with encrypted passwords) and their assigned roles.
- **Role-Based Access Control (RBAC):** The application UI and features will change based on the logged-in user's role:
  - **Sr. Project Manager:** The standard user. Can only see and manage their own assigned projects. Can "Sync" their data.

    - **Principal Engineer:** A technical expert role. Can view projects across all PMs and access specialized technical reports (e.g., RFT date tracking, reports by "Circuit" or "IEN" type).
    - **Associate Director (Admin):** Your role. Has a "Team View" to see all projects for their direct reports. Can access the Admin Control Panel to manage users, formulas, and run the master data import.
    - **Director:** The executive role. Has a high-level portfolio dashboard showing KPIs rolled up by Associate Director and Program.
- **Hierarchy:** The master_users.db will include a Reports_To_ID column to link each user to their manager, automating the creation of team-based reports.

4.0 Data Synchronization (The "Inbox/Processor" Model)

This is the core "safe way" process that allows all local databases to sync to one master database without corruption.

1. **PM Works Locally:** A Sr. PM adds 5 new projects and updates 10 notes on their fast, local my_projects.db.
2. **PM Clicks "Sync":** The app bundles *only those 15 changes* into a small, unique **JSON file** (e.g., sync_JaneDoe_20251023.json). This file is then copied to the G:\PM_Tool\SYNC_INBOX\ folder.
3. **Admin Clicks "Import Data":** At any time, you (as Associate Director) can click "Import Data." Your app will:
    - Get an exclusive lock on the master_projects.db on the G: drive.
    - Open each JSON file in the "Inbox."
    - Safely merge the changes (updates, new projects) into the master database.
    - Move the processed JSON files to an "Archive" folder for auditing.
    - Release the lock.

This one-way data flow makes simultaneous use safe and reliable.

5.0 Core Data Model & Features

The database schema is based on PMI standards and customized for Verizon.

- **Programs Table:** The parent container for high-level initiatives.
- **Projects Table:** The core table for all projects. Includes a program_id (to link to Programs), pm_id (to link to a user), and project_type_id.
- **Project_Types Table:** A list of categories managed by you (e.g., "BAR," "Circuit," "IEN," "BAU," "Special Project - NFL"). This ensures data is clean for filtering.
- **Work_Packages Table:** Tasks, milestones, and deliverables for each project (e.g., "RFT Date").
- **KPI_Snapshots Table:** A historical log of metrics (e.g., Budget, Schedule, On-Time %) taken monthly/quarterly. This is the engine for all performance review reports.
- **config.db (Business Logic):** A central database on the G: drive, editable *only by you*. It stores all **SLA formulas** and **KPI calculations** as text. The local apps read this file at

launch, allowing you to *change business logic for everyone without re-coding the app*.

6.0 User Interface (UI) & Key Workflows

- **"My Dashboard":** The PM's home screen. Shows their personal KPI summary, a "My Tasks" list, and a fully searchable list of *their* 500+ projects.
- **"New Project" Stub:** A PM can create a new project with minimal data (Project ID, Name, Type). All other details can be filled in later.
- **Role-Based Reports:** A dedicated "Reports" tab that shows a different dashboard for each role, from PM-level 6-month eval reports to Director-level portfolio financial summaries.
- **Spreadsheet Importer:** A one-time tool for your Admin panel. It will allow you to open an existing Excel tracker, **map your spreadsheet columns to the new database fields**, and run a bulk import to populate the master_projects.db.

7.0 Artificial Intelligence (AI) Sub-System

The AI will be 100% secure, private, and will learn over time.

- **Local-First AI:** The AI model (from Hugging Face) will be downloaded and run *on the user's local machine*. No data ever leaves the computer, ensuring 100% security.
- **AI Assistant Window:** A dedicated tab with:
  1. **AI Chatbot:** Allows users to ask questions in plain English ("Summarize my 'At Risk' BAR projects").
  2. **AI Hints:** A panel that provides proactive "Next Step" suggestions.
- **AI Knowledge Base:** You will have an admin button to "Run AI Indexer." This will scan read-only copies of files in a specified G: drive folder (PDFs, Visio drawings, spreadsheets) and save the text into a local, searchable ai_knowledge_base.db. The AI can then answer questions *about* those documents.
- **AI Learning Model:** PMs can "thumbs up/down" AI answers. This feedback is safely sent to an **"AI Feedback Inbox"** (as JSON files). You can use this data to retrain and improve the AI model, then place the new model file on the G: drive for everyone to download.

8.0 Full Technical Stack

- **Programming Language: Python** (v3.10+)
- **Database: SQLite 3** (via Python's built-in sqlite3 library).
- **User Interface (UI): Streamlit** (for building the data-heavy dashboards rapidly in pure Python).
- **Data/Import/KPIs: Pandas** (for the spreadsheet importer and running KPI calculations).
- **AI Engine: Hugging Face transformers** & **PyTorch** (for the local on-device AI).
- **Packaging: PyInstaller** (to bundle the entire Python app, AI models, and libraries into a single .exe file).

9.0 Implementation Roadmap

1. **Phase 1 (MVP - Core Tool):** Build the login system, the local database schema, the main

PM dashboard (with project create/edit), the "safe sync" (Inbox) model, and the Spreadsheet Importer.
2. **Phase 2 (Reporting & Admin):** Build the KPI_Snapshots table, the role-based "Reports" tab, and your complete Admin Control Panel (user management, SLA/KPI formula editor).
3. **Phase 3 (AI Assistant):** Integrate the local AI model, the chatbot window, and the AI Knowledge Base indexer/search.

AI Prompt for Phase 1 (MVP) Development

Here is a prompt you can use in a tool like Visual Studio Code (with an AI assistant) to generate the *initial* code for Phase 1 of this project.

Prompt:

Generate a multi-file Python application for a "Local-First" Project Management Tool. This is Phase 1 (MVP) of the project.

Core Technologies:

- **Language:** Python
- **UI:** `Streamlit`
- **Database:** `sqlite3` (built-in)
- **Data Handling:** `Pandas`

File Structure: Create the following file structure:

- `main_app.py` (The Streamlit UI)
- `database.py` (Functions to handle all SQLite operations)
- `auth.py` (Functions for login, user management)
- `importer.py` (Functions for the spreadsheet import)
- `README.md` (Instructions)

Detailed Requirements for Each File:

1. `database.py`:

- Create a function `init_database()` to create all necessary tables in a `master_projects.db` and a `master_users.db`.
- **`master_users.db` Schema:**

    - `users` table: `user_id` (INTEGER PRIMARY KEY), `username` (TEXT), `password_hash` (TEXT), `role` (TEXT, e.g., 'PM', 'Associate Director').
- **`master_projects.db` Schema:**
    - `programs` table: `program_id` (INTEGER PRIMARY KEY), `name` (TEXT).
    - `project_types` table: `type_id` (INTEGER PRIMARY KEY), `name` (TEXT).
    - `projects` table: `project_id` (INTEGER PRIMARY KEY), `name` (TEXT), `ccr_nfid` (TEXT), `program_id` (INTEGER, FOREIGN KEY), `project_type_id` (INTEGER, FOREIGN KEY), `pm_id` (INTEGER, FOREIGN KEY), `status` (TEXT), `phase` (TEXT).
- Create CRUD functions for projects: `get_projects_by_pm(pm_id)`, `create_project(...)`, `update_project(...)`.
- Create functions for populating initial data (e.t., `Project_Types` like 'BAR', 'Circuit').

2. `auth.py`:

- Create a `hash_password(password)` function.
- Create a `check_password(hashed_password, password)` function.
- Create a `login_user(username, password)` function that queries `master_users.db` and returns a user's role or None.
- Create a `create_user(username, password, role)` function (for admin use).
- Use `sqlite3` to connect to `master_users.db`.

3. `importer.py`:

- Create a function `import_from_spreadsheet(file_path, column_mapping)`.
- This function must use `pandas` to read the spreadsheet (`file_path`).
- It should accept a `column_mapping` dictionary (e.g., `{'Project Name': 'name', 'Project ID': 'ccr_nfid'}`).
- It should loop through the Pandas DataFrame, use the mapping, and call the `database.create_project()` function for each row.

4. `main_app.py` (The Streamlit UI):

- Use Streamlit's session state (`st.session_state`) to manage user login.
- **Login View:** If `st.session_state.logged_in` is False, show a login form. Use the `auth.login_user()` function.
- **Main Dashboard View:** If `st.session_state.logged_in` is True:
    - Show a sidebar (`st.sidebar`) with navigation: "My Dashboard," "New Project," "Import," and "Sync."
    - **"My Dashboard":** Display projects from `database.get_projects_by_pm()` in a `st.dataframe`. Add a "New Project" button.
    - **"New Project" Page:** Show a `st.form` to create a new project. It should only require "Project ID (CCR/NFID)," "Project Name," and "Project Type" (as a dropdown).

- **"Import" Page:** (Visible to 'Associate Director' only). Use `st.file_uploader` to upload an Excel file. Display the spreadsheet columns and allow the user to create the `column_mapping` (e.g., using `st.selectbox` for each column). On "Run Import," call `importer.import_from_spreadsheet()`.
- **"Sync" Page:** (Visible to 'PM' only). Add a "Sync My Data" button. For this MVP, this button should just simulate the sync by:
    1. Getting all *new* or *updated* projects for the user.
    2. Creating a JSON file (e.g., `sync_USER_DATE.json`) and saving it to a folder named `SYNC_INBOX`.
    3. Display a "Sync Complete" message.
- **"Process Inbox" Feature:** (Visible to 'Associate Director' only, on the Import page). A button to "Process Sync Inbox." This function should:
    1. List all `.json` files in the `SYNC_INBOX` folder.
    2. Read each file, parse the JSON, and call `database.create_project()` or `database.update_project()`.
    3. Move the processed file to an `ARCHIVE` folder.