1. Overview

This document outlines the high-level system architecture for the **Minimum Viable Product (MVP)** of the "Site-Steward" project. The design is based on the provided proposal and the strategic selection of a **Flask API** backend and a **Streamlit** frontend.

The primary MVP objectives are to validate the two core business propositions:

1. **Reduce Asset Loss** via QR code tracking.
2. **Mitigate Compliance Risk** via document expiry alerts.

The architecture is designed for rapid development, clear separation of concerns, and low initial cost, while establishing a foundation for future scaling.

2. Core Architectural Principles (MVP)

Our design will be a **decoupled, service-oriented architecture**. This is crucial for an MVP, as it allows frontend and backend teams to work in parallel.

1. **Stateless API (Flask):** A central Flask application will serve as the "headless" backend. It will handle all business logic, database interactions, and authentication. It will *only* serve JSON.
2. **Dual-Interface Frontend (Streamlit):** We will develop *two* distinct Streamlit applications that consume the same Flask API. This directly targets our two key user personas.
    - **Admin Portal (Desktop):** For the "Admin (Office)" persona to manage setup, onboarding, and reporting.
    - **Field Mobile App (Mobile-Web):** A lightweight, task-oriented app for the "Site Foreman (Field)" persona.
3. **Managed Database:** A single PostgreSQL database will be the source of truth.
4. **Asynchronous Services:** A simple, scheduled script will handle background tasks (compliance alerts).

This decoupled approach enforces **Single Responsibility (SRP)** and **Dependency Inversion (DIP)** from day one.

3. System Component Breakdown

3.1. Backend API (Flask)

This is the system's core. It will be built with Flask, SQLAlchemy, and Flask-Smorest for rapid, documented API development.

- **Framework:** Flask
- **Database ORM: SQLAlchemy** will be used to map our database tables to Python classes. This directly fulfills the proposal's goal of modeling Assets, Projects, and Compliance Documents as objects.
- **API Structure:** We will use **Flask-Smorest** to build clean, RESTful endpoints that automatically generate OpenAPI (Swagger) documentation. This is vital for

frontend/backend synchronization.

- **Authentication: JSON Web Tokens (JWT)**. An endpoint (/api/login) will return a token. This token will be stored in Streamlit's st.session_state and passed in the Authorization header of all subsequent API requests.
- **File Handling (MVP):** To simplify, PDF uploads will be stored on the local filesystem of the API server, and the file path will be saved in the database. (Note: This will be migrated to a cloud storage bucket in post-MVP scaling).

Key MVP API Endpoints:

Method	Endpoint	Description
POST	/api/login	Authenticates a user, returns a JWT.
GET / POST	/api/assets	Get all assets; create a new asset.
GET	/api/assets/<asset_id>	Get a single asset's details (for QR scan).
POST	/api/assets/<asset_id>/move	Moves an asset to a Project.
GET / POST	/api/subcontractors	Get all subs; create a new sub.
POST	/api/subcontractors/<sub_id>/document	Uploads a new compliance doc (PDF) and expiry date.
GET	/api/projects/<project_id>/compliance	Gets the simple [GREEN]/[RED] compliance status for all subs on a project.

3.2. Frontend: Admin Portal (Streamlit)

This is the "home base" for the office Admin.

- **Technology:** Streamlit
- **Authentication:** A login page will use st.text_input for credentials. On successful API login, the JWT is stored in st.session_state.
- **Key MVP Pages:**
  - **Asset Management:** A page with st.form to add new assets. This page will call the

qrcode library to generate and display a QR code (st.image) for the admin to print.
    - **Subcontractor Management:** A simple CRUD interface (Create/Read/Update/Delete) for the subcontractor directory.
    - **Compliance Upload:** A page using st.file_uploader and st.date_input to upload new insurance PDFs and set their expiry dates.
    - **Project Hub:** A dashboard where the Admin can select a project and see the compliance status.

3.3. Frontend: Field Mobile App (Streamlit)

This app must be simple, fast, and prove the value of mobile scanning.

- **Technology:** Streamlit
- **Primary Technical Risk: Mobile Scanning**. Streamlit does not have a native QR scanner.
- **MVP Solution:** We will use the **streamlit-webrtc** component. This component can access the phone's camera. We will integrate the pyzbar library to process the video stream and detect QR codes. This is the most complex part of the MVP and must be prototyped first.
- **User Flow (Foreman):**
    1. Foreman logs in.
    2. Main page shows two buttons: "Scan Asset" and "View Project Compliance."
    3. **Scan Flow:** "Scan Asset" opens the camera page. On a successful scan, the app calls GET /api/assets/<asset_id> and displays the details. A "Move" button will allow the Foreman to select a project from st.selectbox and call the API.
    4. **Compliance Flow:** "View Project Compliance" shows a project list. Selecting one calls GET /api/projects/<project_id>/compliance and displays the [RED] / [GREEN] status list.

3.4. Asynchronous Services (Compliance Alerts)

This must be a minimal implementation for the MVP.

- **Solution:** A single Python script (check_expiry.py) that will be run once per day via a **Cron Job**.
- **Logic:**
    1. Connects directly to the PostgreSQL database (using SQLAlchemy).
    2. Queries for all ComplianceDocuments expiring within 30 days.
    3. For each result, it will use Python's built-in smtplib to send an email via a standard Gmail/SMTP account. (Note: This will be upgraded to a dedicated email service like SendGrid post-MVP).
- **Rationale:** This implements the **SRP** logic (separated from the API) with minimal setup cost.

4. MVP Deployment

- **API & Frontends:** All three Python applications (Flask API, Admin App, Field App) will be **containerised using Docker**. They can be run locally using Docker Compose or

deployed individually to a simple PaaS (like Heroku) or a single VM for the MVP.

- **Database:** A managed **PostgreSQL** instance (e.g., Heroku Postgres, AWS RDS Free Tier) to avoid database management overhead.