**Project Title: HydroPulseAI – Smart Water Resource Management System**

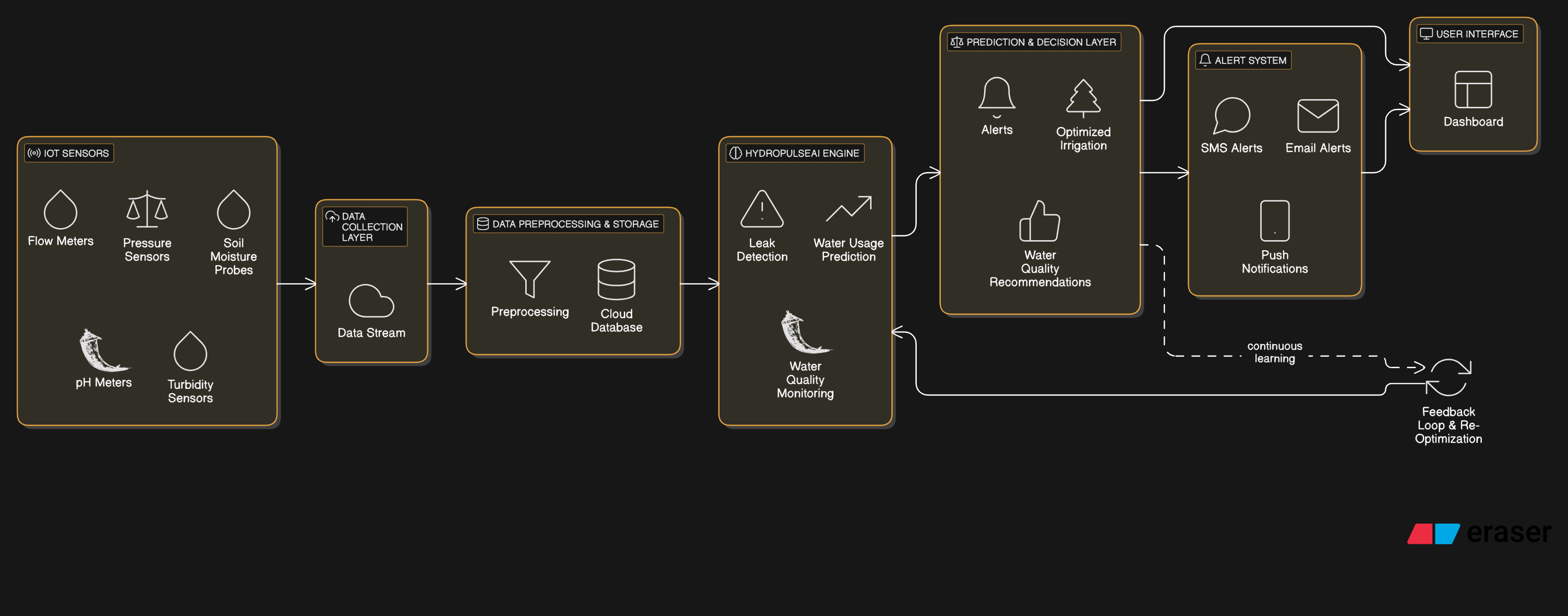
**DOC Version 1.1:**



**Prepared By :**

**A S Saicharan**

**SYSTEM WORKING MODEL :**



**🎯 Project Objective:**

To monitor, detect, and manage water resources in real-time using AI-powered IoT sensors, intelligent alert systems, and a Flutter-based app. The system should classify alerts into severity levels and take automated actions where necessary.

**System Overview & Architecture**

Based on the diagrams and your notes, here's the refined modular breakdown:

**1. IoT Sensors**

* Flow Meters
* Pressure Sensors
* Soil Moisture Probes
* pH Meters
* Turbidity Sensors
* Fire Sensors
* Door Exit Sensors
* Smart Fire Extinguisher Trigger
* Water Level Sensor

These feed data into:

**2. Data Flow Pipeline**

| **Layer** | **Component** | **Role** |
| --- | --- | --- |
| **A. Data Collection** | IoT Device → Gateway → Cloud | Collect and push real-time sensor data |
| **B. API Layer** | Flask/RestAPI (Python) | Three APIs: LeakAPI, UsageAPI, QualityAPI |
| **C. Data Storage** | MySQL / PostgreSQL | Store master and transactional data |
| **D. Alert System** | Twilio / SMTP / Call API | Trigger SMS, email, or call |
| **E. App Layer** | Flutter | User dashboard and control panel |

**API Layer Design**

Use **FastAPI** for high performance and async support.

**✅ 1. LeakAPI**

* Inputs: Pressure, Flow, Water Level
* Logic: If pressure drops and flow spikes → **Leak Detected**
* Output: status: RED/YELLOW, timestamp, location

**✅ 2. UsageAPI**

* Inputs: Historical Usage + Real-time usage
* Logic: If current usage > 2x average → **Overusage Alert**
* Output: status: YELLOW, timestamp, user\_id

**✅ 3. QualityAPI**

* Inputs: pH, Turbidity
* Logic: Outside ideal range? → Trigger alert
* Output: status: ORANGE, pH, turbidity, timestamp

## Database Schema

### 🔹 Master Table – owners

| **Field** | **Type** |
| --- | --- |
| owner\_id | INT (PK) |
| name | VARCHAR |
| email | VARCHAR |
| phone | VARCHAR |
| device\_ip | VARCHAR |
| location | VARCHAR |

### 🔹Transaction Table – events

| **Field** | **Type** |
| --- | --- |
| event\_id | INT (PK) |
| owner\_id | INT (FK) |
| event\_type | ENUM |
| status\_level | ENUM |
| timestamp | DATETIME |
| data\_payload | JSON |

## ****Alert Classification & Action Logic****

| **Severity** | **Trigger** | **Action** |
| --- | --- | --- |
| Yellow | Minor overuse / deviation | Send SMS or Email |
| Orange | Medium anomaly | Missed Call to Owner |
| Red | Severe issue / fire / leak | IoT device triggers auto actions (Fire extinguisher / Valve close / Alarm) |

**Mobile App (Flutter) Features**

**Pages:**

* **Login / Register**
* **Dashboard**
  + Real-time sensor feed
  + Current water usage
  + Alert history
* **Alerts**
  + Yellow/Orange/Red alerts with timestamp
* **Settings**
  + Notification preferences
  + Update contact info
* **Map View**
  + Reservoir locations with statuses

**Implementation Steps**

**🔹 Phase 1: Setup REST API**

Create Flask App

Define /leak, /usage, /quality

Store data in MySQL

Trigger alerts via alert\_handler.py

**🔹 Phase 2: Mobile App (Flutter)**

Setup dashboard

Connect via http.post()

Show alert type (Yellow, Orange, Red)

Push notifications (Firebase or manual)

**🔹 Phase 3: IoT Integration**

Sensors to NodeMCU/RPi

Send data via HTTP POST

On red, auto-trigger extinguisher/door sensor

**Functionalities:**

* Modular APIs
* Central database logging
* AI-based alert decision system
* Web based application/ User login Analytics/MIS Reports/Dashboard
* User-friendly mobile UI
* End-to-end automation (IoT loop)

**Process:**

* **Integrate with Database** – Save every POST request into the transaction table with timestamp.
* **Connect to Flutter App** – Use http package in Flutter to send sensor data to these APIs.
* **Trigger IoT Devices** – For red alerts, trigger an actuator via NodeMCU or Raspberry Pi endpoint.
* **Secure APIs** – Add API key/token auth later for safety.

## Tech Stack

| **Component** | **Technology** |
| --- | --- |
| API Framework | Python + Flask (REST API) |
| Mobile App | Flutter |
| Database | MySQL (Master & Transaction Tables) |
| Hosting | Render / Railway / Local server |
| Alert System | Twilio (SMS/Call) + SMTP (Email) |
| IoT Control | NodeMCU / Raspberry Pi |

## REST API Workflow

1. IoT sends data to /api/leak, /api/usage, or /api/quality

2. Data is stored in Transaction DB with timestamp

3. Based on logic:

- Yellow: SMS/Email

- Orange: Missed Call

- Red: IoT Action (Auto Triggered)

4. Alerts triggered via alert\_handler

5. Dashboard & mobile app show the results

FILE STRUCTURE:  
HydroPulseAI/

├── app.py # Main Flask app file

├── config.py # Configuration settings (DB URI, etc.)

├── db.py # Database initialization

├── models.py # Database models for Owner and Event tables

├── alert\_handler.py # Handles SMS, email, IoT triggers based on severity

├── apis/ # API folder for modular endpoints

│ ├── \_\_init\_\_.py # API initialization

│ ├── leak\_api.py # Leak detection API

│ ├── usage\_api.py # Usage monitoring API

│ └── quality\_api.py # Quality monitoring API

├── utils/ # Utilities folder for reusable functions

│ ├── sms.py # SMS alert utility (Twilio integration)

│ ├── email\_alert.py # Email alert utility (SMTP/SendGrid integration)

│ └── iot\_trigger.py # IoT action utility

├── iot/ # IoT integration folder

│ ├── nodemcu\_code.ino # NodeMCU script for sending HTTP requests

│ └── rpi\_script.py # Raspberry Pi script for IoT control

├── frontend/ # Flutter app folder for UI development

│ ├── lib/ # Main app logic

│ │ ├── main.dart # Flutter main file

│ │ └── screens/ # Screens folder for modular pages

│ │ ├── home\_screen.dart # Home screen for sending data/alerts

│ └── pubspec.yaml # Flutter package manager

└── sql/ # SQL scripts folder

├── create\_tables.sql # SQL code for creating tables