PrecisionPulse - Document of Understanding

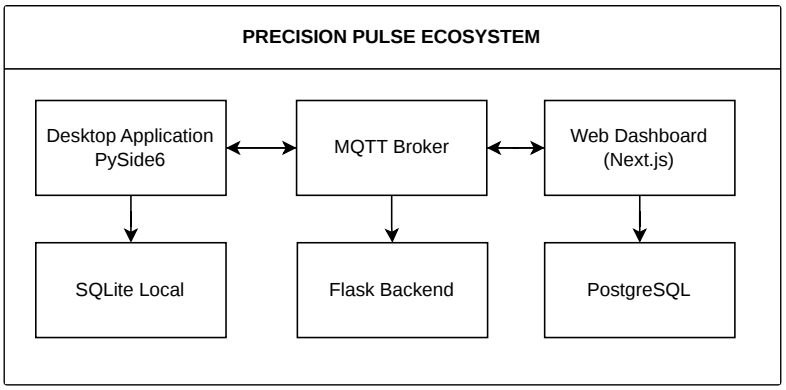
Purpose:

Real-time telemetry streaming and distributed data synchronization.

1. System Architecture:

Decentralized, MQTT-based, Offline-resilient

1.1 Three-Tier Architecture:



1.2 Component Breakdown:

Component	Technology	Purpose	Repository
Web Frontend	Next.js, TypeScript, Tailwind CSS	User interface for monitoring telemetry	PrecisionPulse-Frontend
Backend API	Python Flask, Socket.IO	REST API, MQTT bridge, real-time relay	PrecisionPulse-Backend
Desktop Client	PySide6, Python	Data collection, local storage, MQTT publisher	PrecisionPulse-Desktop
Message Broker	MQTT	Low-latency bi-directional communication	
Web Database	PostgreSQL	Central user management, historical data	

Local Database	SQLite	Offline buffering, credential mirror	

2. Core Functionality:

Real-time telemetry streaming and distributed data synchronization.

2.1 Distributed Identity & Management:

2.1.1 Credential Storage

- **No Central Database:** Credentials exist in both PostgreSQL (web) and SQLite (desktop)
- **Unified Authentication:** Same email/password works on both platforms
- **Synchronization:** MQTT triggers credential updates across systems

2.1.2 Role-Based Access Control (RBAC)

Roles:

- **admin** - Full system access
- **user** - Read-only telemetry access

Permissions:

Permission	Admin	User
View Dashboard	✅	✅
View Telemetry Data	✅	✅
View Profile	✅	✅
Change Own Password	✅	✅
Manage Users	✅	❌
Create Users	✅	❌
Edit User Role	✅	❌
Set Initial Password	✅ (create only)	❌
Delete Users	✅ (except self)	❌
Toggle User Status	✅	❌
Command & Control	✅	❌

Force Sync	✅	❌
Update Config	✅	❌
Configure Telemetry (Desktop)	✅	❌

2.2 Real-Time Telemetry Streaming

2.2.2 Telemetry Data Types

- **Integers:** Long/Double values (e.g., counts, IDs)
- **Decimals:** Float values (e.g., temperature, pressure)
- **Booleans:** Status flags (e.g., active/inactive)

2.2.3 Real-Time Features

- **Zero-Refresh UI:** WebSocket updates without page reload
- **Sub-Second Latency:** MQTT ensures fast delivery
- **Live Status Indicator:** Green (connected) / Red (disconnected)
- **Share-Market Style:** Continuous streaming updates

3. Technical Implementation

Purpose:

Implementation details of Web, Backend, Desktop, and MQTT communication layers.

System Architecture:

Next.js (Frontend) + Flask (Backend) + PySide6 (Desktop) + MQTT (Messaging Layer)

3.1 Web Frontend (Next.js)

3.1.1 Pages & Routes

Route	Access	Purpose
`/login`	Public	User authentication
`/dashboard`	Authenticated	Real-time telemetry display
`/profile`	Authenticated	User profile & password change

`/users`	Admin only	User management (CRUD)

3.1.2 Key Features

- **JWT Authentication**: Token-based auth with 24h expiry
- **Multi-Layer RBAC**: Middleware + Component guards
- **Protected Routes**: Server-side validation
- **Permission-Based UI**: Conditional rendering
- **Real-Time Updates**: Socket.IO client integration

3.1.3 Security Layers

1. **Middleware (Server-side)**
   - Token validation
   - Role checking
2. **ProtectedRoute Component**
   - Page-level access control
3. **RBACGuard Component**
   - Element-level permission checks

3.2 Backend API (Flask)

3.2.1 Responsibilities

- REST API endpoints for authentication
- MQTT subscriber (receives from desktop)
- Socket.IO server (broadcasts to web)
- PostgreSQL database operations
- Credential sync orchestration

3.3 Desktop Application (PySide6)

3.3.1 Features

- **Login Window**: Same credentials as web
- **Telemetry Dashboard**: Real-time data collection view
- **Settings Panel**: Configuration (admin only)
- **Sync Status**: Visual indicator of connection state
- **Background Service**: Continuous data collection

4. User Management Rules

4.1 User Creation

- Only admins can create users
- Admin sets initial password during creation
- Admin assigns role (admin/user)
- No self-registration allowed
- New user credentials synced to desktop via MQTT

4.2 User Editing

- Admin can change user role anytime
- Admin cannot change password after creation
- Users change own password via Profile page
- Admin cannot delete themselves
- Admin can toggle user active/inactive status

4.3 Password Management

- **Creation**: Admin sets initial password
- **Change**: User changes via Profile page
- **Hashing**: bcryptjs with salt rounds
- **Validation**: Minimum 6 characters
- **Sync**: Password changes trigger MQTT sync

5. Data Flow Scenarios

5.1 Normal Operation:

- Desktop collects sensor data every 1 second
- Desktop publishes to MQTT topic "telemetry/data"
- Flask backend subscribes and receives data
- Backend emits to Socket.IO clients
- Web dashboard updates UI in real-time
- Backend stores in PostgreSQL for history

5.2 Offline Operation

- Desktop loses internet connection
- Desktop detects MQTT broker unreachable
- Desktop redirects data to SQLite buffer table

- Desktop continues collecting data locally
- Connection restored
- Desktop reads buffer table (oldest first)
- Desktop publishes buffered data to MQTT
- Desktop waits for acknowledgment
- Desktop deletes synced records from buffer
- Desktop resumes live streaming

5.3 User Creation & Sync

- Admin logs into web dashboard
- Admin navigates to /users page
- Admin clicks "Add User"
- Admin fills form (name, email, password, role)
- Backend creates user in PostgreSQL
- Backend publishes to MQTT "users/sync"
- Desktop receives sync message
- Desktop inserts/updates user in SQLite
- New user can now login on both platforms

6. Security Implementation

6.1 Authentication

- **Algorithm**: JWT with HS256
- **Token Storage**: localStorage + HTTP-only cookie
- **Expiration**: 24 hours
- **Refresh**: Manual re-login required

6.2 Password Security

- **Hashing**: bcryptjs (same on web & desktop)
- **Salt Rounds**: 10
- **Validation**: Minimum 6 characters
- **Transmission**: HTTPS/TLS only

6.3 MQTT Security

- **Encryption**: TLS/SSL enabled
- **Authentication**: Username/password

- **Authorization**: Topic-based ACL
- **Payload**: Encrypted sensitive data

6.4 Route Protection

- **Middleware**: Server-side token validation
- **Role Checking**: Admin routes blocked for users
- **Component Guards**: Client-side double-check
- **Automatic Redirect**: Unauthorized → /login or /dashboard