Resource Monitoring and Management System

Quick Reference Guide - Component 8 Documentation

📋 Document Structure

This document follows the same structure as the Pharmacy Inventory Management System (PharmPal) Component 8 report:

A. Design of Forms & Frontend Screenshots

- Login Screen
- Main Dashboard Screen
- System Detail Screen
- Alert Configuration Form

B. Security and Validation Proof

- Authentication System (JWT)
- Password Hashing (PBKDF2-SHA256)
- Data Integrity (Foreign Keys)
- API Endpoint Protection
- Agent-Server Communication

C. Innovative Experiment

- Automated Agent Deployment
- Real-Time Monitoring Architecture
- Zero-Configuration System

D. Database Structure

- 6 tables with full relationships
- Verification scripts included

🔐 Security Implementation Summary

Security Feature	Implementation	Location
Authentication	JWT Tokens	`backend/app/api/auth.py`
Password Storage	PBKDF2-SHA256 Hash	`backend/app/core/security.py`
Database	SQLite with constraints	`backend/resource_monitor.db`
Validation	Pydantic Schemas	`backend/app/schemas/schemas.py`
SQL Injection	SQLAlchemy ORM	All database queries

👥 Current Users (Verification)

```
Database: backend/resource_monitor.db
Table: users

ID | Email                    | Password Hash                | Status | Created
---|------------------------|----------------------------|--------|------------
1  | nthy2355@gmail.com     | $pbkdf2-sha256$29000$...  | Active | 2025-12-25
2  | Admin1@gmail.com       | $pbkdf2-sha256$29000$...  | Active | 2025-12-27
```

✓ Passwords are securely hashed - original passwords cannot be retrieved

🗄️ Database Tables

```
1. users (2 records)
   ├── id, email, hashed_password, is_active, created_at

2. systems
   ├── id, hostname, ip_address, mac_address
   ├── os_info, cpu_name, memory, disk, gpu
   └── Foreign keys → metrics, alerts, tickets

3. metrics
   ├── id, system_id (FK), timestamp
   └── cpu_usage, memory_percent, disk_usage, network_stats

4. alerts
   ├── id, system_id (FK)
   └── alert_type, severity, message, is_resolved

5. tickets
   ├── id, system_id (FK)
   └── message, status, resolved_at

6. alert_settings
   ├── id, system_id (FK)
   └── cpu_threshold, memory_threshold, disk_threshold
```

📁 Key File Locations

Backend (FastAPI)

```
backend/
├── app/
│   ├── api/
│   │   ├── auth.py              ← Authentication endpoints
│   │   └── endpoints.py         ← Main API endpoints
│   ├── core/
│   │   ├── security.py          ← Password hashing
│   │   ├── alerts.py            ← Alert generation
│   │   └── discovery.py         ← System discovery
│   ├── db/
│   │   └── database.py          ← Database connection
│   ├── models/
│   │   └── models.py            ← SQLAlchemy models (6 tables)
│   └── schemas/
│       └── schemas.py           ← Pydantic validation
└── resource_monitor.db          ← SQLite database
```

Frontend (React)

```
dashboard/
└── src/
    ├── pages/
    │   ├── Login.jsx            ← Login form
    │   ├── Dashboard.jsx        ← Main dashboard
    │   └── SystemDetail.jsx     ← System details
    └── services/
        └── api.js               ← API integration
```

Agent (Python)

```
agent/
├── main.py                      ← Main agent logic
└── gui.py                       ← System tray interface
```

Build System

```
setup_lab.py                    ← Automated IP detection & build
build_lab_agent.bat             ← One-click agent build
```

🚀 Validation Examples

Frontend Validation (React)

```
// Email format validation
const validateEmail = (email) => {
  return /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email);
};

// Required field validation
if (!email || !password) {
  setError("All fields are required");
}
```

Backend Validation (Pydantic)

```
class UserCreate(BaseModel):
    email: str  # Must be string
    password: str  # Must be string

class MetricCreate(BaseModel):
    cpu_usage: float  # Must be float (0-100)
    memory_percent: float  # Auto-validated
    process_count: int  # Must be integer
```

Database Validation (SQLAlchemy)

```
class User(Base):
    email = Column(String, unique=True, nullable=False)
    # ↑ Database enforces uniqueness and NOT NULL
```

🔧 Verification Scripts

Run these scripts to verify the implementation:

View Users

```
python inspect_db.py
```

Detailed User Data

```
python view_all_user_data.py
```

Open Database GUI

```
open_database.bat
```

Extract PDF Documentation

```
python read_pdf.py
```

💡 Innovative Features

1. Automated IP Detection

```
# setup_lab.py
def get_local_ip():
    # Automatically detects server IP
    # Embeds it into agent during build
    # Zero manual configuration needed!
```

2. Foreign Key Cascade

```
# When a system is deleted, all related data is automatically removed
class Metric(Base):
    system_id = Column(Integer, ForeignKey("systems.id"))
    system = relationship("System", back_populates="metrics",
                         cascade="all, delete-orphan")
```

3. Real-Time Monitoring

- Agent sends metrics every N seconds
- Dashboard polls for updates
- Automatic alert generation
- System discovery

📊 Comparison with PharmPal Project

Feature	PharmPal	Resource Monitor
Frontend	Flutter	React.js
Backend	FastAPI	FastAPI
Database	PostgreSQL	SQLite
Auth	JWT + Argon2	JWT + PBKDF2
Innovation	AI Chatbot	Auto-deployment
Multi-user	Yes (per-user inventory)	Yes (shared monitoring)

📖 Documentation Files

1. **Component_8_Resource_Monitoring_System.md** (This report)
2. Complete technical documentation
3. Security proof
4. Validation examples

5. Database schema

6. **HOW_TO_VIEW_USER_DATA.md**

7. Guide for accessing user data
8. Multiple access methods

9. Quick reference

10. **security_proof.md** (Existing)

11. Detailed security analysis
12. Vulnerability assessment

✅ Checklist

- [x] Multi-user authentication (JWT)
- [x] Password hashing (PBKDF2-SHA256)
- [x] Database with foreign keys
- [x] Frontend validation
- [x] Backend validation (Pydantic)
- [x] Database constraints
- [x] SQL injection prevention (ORM)
- [x] XSS prevention (React escaping)
- [x] Automated deployment
- [x] Real-time monitoring
- [x] Alert system
- [x] Comprehensive documentation

Report Created: December 27, 2025
Project: Resource Monitoring and Management System
Repository: MadeNavaneeth/Resource-Monitoring-and-Management-System

---

For complete details, see: `Component_8_Resource_Monitoring_System.md`