Backend Development Tasks

Monitoring Dashboard API

Project: Tazrout Dashboard

February 14, 2026

Overview

This document outlines your responsibilities as the **Backend and Database Developer** for the Monitoring Dashboard project. The frontend developer has completed the mockups and design system. Your task is to build the FastAPI backend that serves the Flutter desktop application.

Important Notes

- Start with **mock data** - the data model is not finalized yet
- We will refine the data structure together in 2-5 days
- Mock data should be **realistic and match the mockups**
- All mockups are in `design/mockups/` folder in the repository
- Check Figma link in `design/FIGMA_LINKS.md`

1 Phase 2: Backend Foundation

1.1 Project Setup

1. Navigate to `backend/` directory
2. Create Python virtual environment:

```
python -m venv venv
source venv/bin/activate  # Windows: venv\Scripts\activate
```

3. Install dependencies from `requirements.txt`
   Note : the file is empty now so if you have the rquirements add them.
4. Verify FastAPI runs: `uvicorn app.main:app --reload`

1.2 Configuration Files

1. Create a file called `.env.example` in the `backend` folder
2. Put this content inside it, and you can add,remove or adjust it as you want. It is just a template.

.env Configuration

```
# BACKEND ENVIRONMENT VARIABLES

# APPLICATION SETTINGS
APP_NAME=Monitoring Dashboard API
APP_VERSION=1.0.0
APP_ENV=development

# SERVER CONFIGURATION
API_HOST=0.0.0.0
API_PORT=8000
API_RELOAD=true


# LOGGING
LOG_LEVEL=debug
LOG_FILE=logs/app.log


# CORS (Cross-Origin Resource Sharing)
CORS_ORIGINS=["http://localhost:3000", "http://localhost:8080"]


# DATABASE CONNECTION

# DEVELOPMENT (using mock data):
DATABASE_URL=mock://localhost/monitoring_dev
USE_MOCK_DATA=true


DB_POOL_SIZE=5
DB_MAX_OVERFLOW=10
DB_POOL_TIMEOUT=30

# SECURITY
SECRET_KEY=your-secret-key-change-in-production
ACCESS_TOKEN_EXPIRE_MINUTES=30


# EXTERNAL SERVICES

# Add your external service configurations here
EXTERNAL_API_URL=
EXTERNAL_API_KEY=

# MOCK DATA SETTINGS
MOCK_DATA_PATH=mock_data/
MOCK_ZONES_COUNT=5
MOCK_HISTORY_DAYS=30

# DEVELOPMENT

DEBUG=true
ENABLE_DOCS=true
```

3. Copy `backend/.env.example` to `backend/.env`

4. Keep `USE_MOCK_DATA=true` for now
5. Configure `backend/app/config.py` with settings
6. Set up logging in `backend/app/utils/logger.py`

1.3 Core Application Structure

1. Complete `backend/app/main.py`:
   - FastAPI app initialisation
   - CORS middleware configuration
   - API router registration
   - Exception handlers
2. Create health check endpoint: `GET /api/v1/health`
3. Test with Swagger docs at `http://localhost:8000/docs`

1.4 Data Models (Preliminary)

Based on mockups, create these Pydantic models in `backend/app/models/`:
Zone Model (`zone.py`):

- `id`: int
- `name`: str (e.g., "Zone A - Tomatoes")
- `crop_type`: str
- `status`: str ("active", "idle", "error")
- `area`: float (square meters)
- `location`: str (optional)
- `created_at`: datetime
- `updated_at`: datetime

Sensor Reading Model (`sensor_reading.py`):

- `id`: int
- `zone_id`: int
- `moisture_percentage`: float
- `temperature_celsius`: float
- `humidity_percentage`: float
- `battery_percentage`: float
- `is_calibrated`: bool
- `timestamp`: datetime

AI Decision Model (`ai_decision.py`):

- `id`: int
- `zone_id`: int
- `decision`: str (”irrigate”, ”skip”)
- `rationale`: str
- `confidence_score`: float (0.0 to 1.0)
- `timestamp`: datetime
- `executed`: bool
- `sensor_data_snapshot`: dict (optional)

Irrigation Log Model (`irrigation_log.py`):

- `id`: int
- `zone_id`: int
- `water_used_liters`: float
- `duration_minutes`: float
- `start_time`: datetime
- `end_time`: datetime
- `triggered_by`: str (”manual”, ”ai”, ”emergency_stop”)

1.5 Pydantic Schemas

Create request/response schemas in `backend/app/schemas/`:

- `zone_schema.py`: ZoneResponse, ZoneList
- `sensor_schema.py`: SensorReadingResponse, SensorReadingList
- `ai_decision_schema.py`: AIDecisionResponse, AIDecisionList
- `irrigation_schema.py`: IrrigationLogResponse, AnalyticsResponse
- `common.py`: PaginationParams, ErrorResponse

2 Phase 3: Mock Data Generation

2.1 Mock Data Generators

Create realistic data generators in `backend/mock_data/generators/`:

1. Zone Generator (`generate_zones.py`):

- Generate 5-10 zones
- Different crop types (tomatoes, peppers, lettuce, etc.)
- Mix of statuses (active, idle, error)
- Realistic areas (100-500 sq meters)

- Save to `backend/mock_data/zones.json`

2. Sensor Data Generator (`generate_sensors.py`):

- Generate last 30 days of readings
- 15-minute intervals (96 readings per day)
- Realistic patterns:
  - Moisture decreases over time (50-90%)
  - Temperature follows daily cycles (15-35 C)
  - Humidity varies (30-70%)
  - Battery slowly depletes (70-100%)
- Save to `backend/mock_data/sensor_readings.json`

3. AI Decision Generator (`generate_ai.py`):

- Generate 2-4 decisions per day per zone
- Decisions based on moisture levels
- Include realistic rationale text
- Confidence scores 0.7-0.99
- Save sensor snapshots with each decision
- Save to `backend/mock_data/ai_decisions.json`

4. Irrigation Log Generator (`generate_irrigation.py`):

- Match irrigation events to AI decisions
- Water usage: 100-300 liters per event
- Duration: 10-30 minutes
- Show moisture increase after irrigation
- Save to `backend/mock_data/irrigation_logs.json`

2.2 Mock Data Service

Create `backend/app/services/mock_data_service.py`:

- Load JSON files at startup
- Functions to query mock data
- Filter by date ranges
- Pagination support
- Sorting capabilities

3 Phase 4: API Endpoints

3.1 API Router Setup

In `backend/app/api/v1/router.py`:

- Set up APIRouter with prefix `/api/v1`
- Include all endpoint routers
- Add tags for organization

3.2 Zones Endpoints

Create `backend/app/api/v1/endpoints/zones.py`:

1. `GET /api/v1/zones`
    - Return all zones
    - Support pagination (skip, limit)
    - Return ZoneList schema
2. `GET /api/v1/zones/{id}`
    - Return single zone by ID
    - Handle 404 if not found
    - Return ZoneResponse schema
3. `GET /api/v1/zones/{id}/status`
    - Return zone with latest sensor reading
    - Include current irrigation status
    - Include last irrigated timestamp

3.3 Sensor Endpoints

Create `backend/app/api/v1/endpoints/sensors.py`:

1. `GET /api/v1/sensors/readings`
    - Query parameters: zone_id, limit, start_date, end_date
    - Filter mock data accordingly
    - Return SensorReadingList schema
    - Support pagination
2. `GET /api/v1/sensors/latest`
    - Optional zone_id parameter
    - Return latest reading per zone
    - Return dict: {zone_id: reading}

3.4 AI Decisions Endpoints

Create `backend/app/api/v1/endpoints/ai_decisions.py`:

1. `GET /api/v1/ai/decisions`
   - Query parameters: zone_id, start_date, end_date, limit
   - Filter and paginate mock data
   - Return AIDecisionList schema
2. `GET /api/v1/ai/decisions/{id}`
   - Return single decision
   - Include sensor snapshot if available
   - Handle 404 if not found

3.5 Irrigation Endpoints

Create `backend/app/api/v1/endpoints/irrigation.py`:

1. `GET /api/v1/irrigation/logs`
   - Query parameters: zone_id, start_date, end_date
   - Filter mock data
   - Return irrigation log list
2. `GET /api/v1/irrigation/analytics`
   - Query parameter: period (daily, weekly, monthly)
   - Calculate aggregated statistics:
     - Total water used
     - Average per day
     - Peak day and amount
     - Breakdown by zone
     - Daily breakdown (date, amount)
   - Return AnalyticsResponse schema

3.6 Emergency Endpoints

Create `backend/app/api/v1/endpoints/emergency.py`:

1. `POST /api/v1/emergency/stop`
   - Accept confirmation in request body
   - Log emergency stop to JSON file
   - Return success response
   - (Hardware broadcast will be added later)
2. `GET /api/v1/emergency/status`
   - Return current emergency status
   - Check latest emergency log

4 Phase 5: Testing & Documentation

4.1 Unit Tests

In `backend/tests/unit/`:

- Test mock data service functions
- Test utility functions
- Test schema validation

4.2 Integration Tests

In `backend/tests/integration/`:

- Test all API endpoints
- Test with various query parameters
- Test error handling (404, 400, etc.)
- Verify response schemas
- Test pagination

4.3 API Documentation

1. Add docstrings to all endpoint functions
2. Add request/response examples to schemas
3. Test Swagger UI at `/docs`
4. Export OpenAPI schema to `docs/openapi.json`
5. Write `docs/API_DOCUMENTATION.md`:
    - Document all endpoints
    - Include request/response examples
    - Document query parameters
    - Document error codes

4.4 Code Quality

- Run linter: `flake8 app/`
- Run formatter: `black app/`
- Add type hints to all functions
- Add docstrings to all modules
- Ensure test coverage ¿ 80%

5 Phase 13: Containerization (Backend Part)

5.1 Docker Setup

1. Create `backend/Dockerfile`:
   - Multi-stage build
   - Python 3.10-slim base
   - Copy requirements and install dependencies
   - Copy application code
   - Expose port 8000
   - Set CMD to run uvicorn
2. Create `backend/.dockerignore`
3. Test Docker build:

```
docker build -t monitoring-backend backend/
docker run -p 8000:8000 monitoring-backend
```

5.2 Docker Compose

Update `deployment/docker/docker-compose.dev.yml`:

- Define backend service
- Map port 8000
- Mount code volume for hot reload
- Set environment variables

6 Timeline & Milestones

Suggested Timeline

Week 1:

- Days 1-2: Project setup, configuration, basic structure
- Days 3-4: Mock data generation, data service
- Days 5-7: API endpoints implementation

Week 2:

- Days 1-2: Complete all endpoints
- Days 3-4: Testing (unit + integration)
- Days 5-6: Documentation, code quality
- Day 7: Docker setup, review with frontend developer

7 Communication & Collaboration

7.1 Git Workflow

- Work in `backend/` directory only by creating a separate branch called "dashboard-backend".
- Commit frequently with clear messages
- Hint: (use this convention)

```
[INIT] - Initial setup
[ADDED] - New features/files
[UPDATED] - Modify existing
[FIXED] - Bug fixes
[DOCUMENTED] - Documentation
[REFACTORED] - Code restructure
[CONFIGURED] - Configuration changes
[IGNORED] - IGNORING it git ignore
[MERGING] - MERGE BRANCHES
```

- Push to `dashboard-backend-dev`
- Don't modify `frontend/` directory
- Start your structure from the main branch and don't push to the main branch.

7.2 Data Model Meeting

- Schedule meeting in 2-3 days
- Review mockups together
- Finalize field names and types
- Document in `docs/DATA_MODEL.md`
- Adjust mock data if needed

7.3 API Contract

- Once endpoints are ready, share Swagger URL
- I will start integration
- Communicate any breaking changes
- Version API if major changes needed

8 Resources

8.1 Documentation

- FastAPI: `https://fastapi.tiangolo.com/`
- Pydantic: `https://docs.pydantic.dev/`
- Pytest: `https://docs.pytest.org/`

8.2 Repository

- Mockups: `design/mockups/`
- Figma link: `design/FIGMA_LINKS.md`
- Project docs: `docs/`
- Your workspace: `backend/`

9 Questions?

If you have any questions or need clarification:

- Review the mockups in `design/mockups/`
- Check the Figma file (link in repository)
- Ask the frontend developer
- Document decisions in `docs/`

Good luck with the backend development!
Remember: Start with mock data, we'll refine together soon.