Growatt Monitor

Growatt Devices Monitor - System Architecture

Growatt Devices Monitoring System Documentation

May 02, 2025

Version 1.0

Growatt Devices Monitor - System Architecture

1. System Overview

The Growatt Devices Monitor is a web-based application designed to monitor and visualize data from Growatt solar energy devices. It provides real-time monitoring, historical data analysis, visualization tools, and predictive analytics for solar energy production. The system follows a modern web application architecture with a clean separation of concerns between backend, frontend, and data processing components.

2. Architecture Components

2.1 Backend Architecture

2.1.1 Core Application Framework

- **Flask Web Framework**: The application is built using Flask, a lightweight Python web framework that serves as the foundation for handling HTTP requests, routing, and responses.
- **Modular Blueprint Structure**: The application uses Flask Blueprints for modular organization of routes:
- `main_routes.py`: Primary application routes for web pages
- `api_routes.py`: API endpoints for data retrieval
- `data_routes.py`: Endpoints for data collection and management
- `prediction_routes.py`: ML-based prediction services

2.1.2 Data Access Layer

- **API Integration**: The `core/growatt.py` module provides a client for interacting with the Growatt API, handling authentication, data retrieval, and transformations.
- **Database Layer**: The `database.py` module handles database connections, schema management, and data persistence using SQLite.
- **Service Layer**: Services like `plant_service.py` act as intermediaries between the web routes and data storage, implementing business logic.

2.1.3 ML Capabilities

- **Energy Predictor**: The `ml/energy_predictor.py` module implements machine learning capabilities for energy production forecasting based on historical data, seasonal patterns, and environmental factors.
- **Prediction API**: Exposed through dedicated endpoints that allow frontend components to access predictive analytics.

2.2 Frontend Architecture

2.2.1 UI Framework

- **Template System**: Uses Flask's Jinja2 templating with a component-based approach, organized in the `templates/` directory.
- **Base Layout**: `base.html` provides the application shell with common elements and responsive behavior.
- **Components**: Modular UI components in `templates/components/` for reusability:
- General components (navbar, footer)
- Dashboard components (cards, metrics)
- Data visualization (charts, maps)

2.2.2 JavaScript Architecture

- **Alpine.js Framework**: Lightweight JavaScript framework for declarative UI behavior without a build step.
- **Component Management**: `component-loader.js` provides dynamic loading of components with caching.
- **Chart Rendering**: Integration with Chart.js for data visualization in various formats.
- **Interactivity Modules**:
- `map-interaction.js`: Handles interactive maps of solar installations
- `alpine-extensions.js`: Custom extensions to enhance Alpine.js functionality
- App-specific components like `plants-app.js` for managing plant data

2.2.3 Responsive Design

- **TailwindCSS Framework**: Utility-first CSS approach for consistent styling
- **Responsive Breakpoints**: Defined in `base.html` and managed through JavaScript for adaptive layouts
- **Device Detection**: Responsive state object (`window.responsive`) tracks device characteristics and triggers layout adjustments

2.3 Data Flow Architecture

2.3.1 Data Collection

- **Data Collector**: `data_collector.py` handles scheduled data retrieval from Growatt API
- **Cron Integration**: Scheduled tasks for regular data synchronization
- **Manual Collection**: API endpoints for on-demand data collection

2.3.2 Data Storage

- **SQLite Database**: Local storage using SQLite for device data, energy statistics, and weather information
- **Table Structure**:
- Plants: Solar installation information
- Devices: Individual device details linked to plants
- Energy Stats: Production statistics over time
- Weather Data: Environmental conditions

2.3.3 Data Caching

- **Browser-side Caching**: Implemented in frontend components using localStorage
- **Server-side Caching**: Flask-Caching integration for API responses
- **Cache Invalidation**: Both time-based expiration and explicit refresh mechanisms

3. Key Features and Implementations

3.1 Dashboard and Monitoring

- **Real-time Updates**: Dynamic data refresh for current production values
- **System Status Overview**: Displays key metrics, alerts, and performance indicators
- **Device Status Tracking**: Monitoring online/offline status of connected devices

3.2 Data Visualization

- **Energy Yield Charts**: Customizable charts for viewing production data across different time periods
- **Power Flow Visualization**: Interactive representation of energy flow between components
- **Thailand Solar Map**: Geographic visualization of plant locations with status indicators
- **Performance Analytics**: Charts for efficiency, distribution, and other key metrics

3.3 Predictive Analytics

- **Energy Production Forecasting**: ML-based predictions of future energy production
- **Seasonal Adjustments**: Accounting for seasonal variations in production estimates
- **Performance Ratio Analysis**: Comparing actual vs. theoretical output for system evaluation

3.4 Weather Integration

- **Weather Data Correlation**: Integration of weather data with energy production
- **Condition Visualization**: Charts showing relationship between weather and system performance
- **Historical Weather Data**: Stored alongside production data for analysis

4. System Interaction Flow

4.1 Authentication Flow

1. User accesses the application
2. Application checks for existing session
3. If not authenticated, login form is presented
4. Credentials sent to Growatt API via secure endpoint
5. On success, session is created and user is redirected to dashboard

4.2 Data Retrieval Flow

1. Frontend components request data through API endpoints
2. Server-side routes handle requests and validate authentication
3. Service layer retrieves data from local database or external API
4. Data is formatted and returned as JSON
5. Frontend components render visualizations based on received data

4.3 Prediction Flow

1. User requests prediction through UI
2. Frontend sends request to prediction endpoint
3. ML model processes request with appropriate parameters
4. Prediction results returned to frontend
5. Results displayed in charts with confidence indicators

5. Cross-cutting Concerns

5.1 Responsive Design

- Adaptive layouts for mobile, tablet, and desktop devices
- Breakpoint-based component rendering
- Touch-friendly interactions for mobile users

5.2 Error Handling

- Graceful degradation when API is unavailable
- User-friendly error messages
- Automatic retry mechanisms with exponential backoff

5.3 Performance Optimization

- Data caching at multiple levels
- Lazy loading of components
- Throttled API requests
- Client-side data processing where appropriate

5.4 Security Measures

- Secure authentication flow
- Session management
- CORS configuration
- Error output sanitization in production

6. Deployment Architecture

6.1 Development Environment

- Local Flask development server
- Live reload for rapid iteration
- Local database for testing

6.2 Production Deployment

- Containerization with Docker and docker-compose

- Nginx as reverse proxy with SSL termination
- Gunicorn as WSGI server for Flask application
- Persistent volume for database and logs

7. External Integrations

7.1 Growatt API

- Authentication mechanism
- Data retrieval endpoints
- Command interfaces for device control

7.2 Weather Services

- Integration with weather data providers
- Historical weather data correlation
- Weather forecasting for prediction enhancement

8. Future Architecture Considerations

8.1 Scalability Improvements

- Migration to PostgreSQL for larger installations
- Redis for improved caching and session management
- Microservice decomposition for specialized components

8.2 Feature Expansions

- Mobile application with push notifications
- Advanced anomaly detection
- Integration with smart home systems
- Expanded ML capabilities for maintenance prediction

9. Technology Stack Summary

- **Backend**: Python, Flask, SQLite
- **Frontend**: HTML, Alpine.js, TailwindCSS, Chart.js

- **Data Visualization**: Chart.js, SVG interactive maps
- **Machine Learning**: NumPy, custom prediction models
- **Deployment**: Docker, Nginx, Gunicorn