

B.M.S COLLEGE OF ENGINEERING BENGALURU
Autonomous Institute, Affiliated to VTU



OOMD Mini Project Report on
Solar Panel Monitoring and Management System (SPMMS)

Submitted in partial fulfillment for the award of degree of

Bachelor of Engineering
in
Computer Science and Engineering

Submitted by:

Chiraiya Sethiya(1BM23CS080)
Himika Kakhani (1BM23CS112)
Hemanshu Dhiman (1BM23CS109)



Faculty In charge
Vikranth B.M
Assistant Professor
Department of Computer Science and Engineering
B.M.S College of Engineering
Bull Temple Road, Basavanagudi, Bangalore 560 019
2022-2023

B.M.S COLLEGE OF ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



DECLARATION

We, Chiraiya Sethiya(1BM23CS080), Himika Kakhani(1BM23CS112) and Hemanshu Dhiman(1BM23CS109), students of 5th Semester, B.E, Department of Computer Science and Engineering, BMS College of Engineering, Bangalore, hereby declare that, this OOMD Mini Project entitled "Solar Panel Monitoring and Management System (SPMMS)" has been carried out in Department of CSE, BMS College of Engineering, Bangalore during the academic semester August - January 2026. I also declare that to the best of our knowledge and belief, the OOMD mini Project report is not from part of any other report by any other students.

Signature of the Candidate

Chiraiya Sethiya (1BM23080)
Himika Kakhani (1BM23CS112)
Hemanshu Dhiman (1BM23CS109)

BMS COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND

ENGINEERING



CERTIFICATE

This is to certify that the OOMD Mini Project titled "**Solar Panel Monitoring and Management System (SPMMS)**" has been carried out by Chiraiya Sethiya(1BM23CS080), Himika Kakhani(1BM23CS112) and Hemanshu Dhiman(1BM23CS109) during the academic year 2025-2026.

Signature of the Faculty in Charge

Vikranth B.M

Assistant Professor

Table of Contents

| Sl No. | Title | Page No. |
|---------------|------------------------------------|-----------------|
| 1 | Problem Statement | 1-2 |
| 2 | Software Requirement Specification | 3-10 |
| 3 | Class Diagram | 8-9 |
| 4 | State Diagram | 13-14 |
| 5 | Interaction Diagram | 12-24 |

1. INTRODUCTION

In recent years, renewable energy sources have become crucial for addressing global energy challenges and reducing environmental impact. Among them, **solar power** has emerged as one of the most promising and widely adopted clean energy alternatives. However, the performance and reliability of solar panel installations depend on continuous monitoring, timely maintenance, and efficient energy management.

Traditional solar setups often lack the ability to track performance metrics in real time or predict potential failures. This limitation can lead to reduced efficiency, higher maintenance costs, and energy losses. The **Solar Panel Monitoring and Management System (SPMMS)** aims to provide a comprehensive solution to these challenges by leveraging **IoT technology, cloud computing, and data analytics**. The system continuously collects data from solar panels, analyzes performance parameters, detects anomalies, and predicts maintenance needs — ensuring optimal energy generation and system longevity.

Problem Statement:

Although solar energy systems are widely adopted, many existing setups face key challenges such as:

- Lack of **real-time monitoring** of energy generation and system performance.
- Inability to **detect faults or inefficiencies** in time, leading to reduced energy output.
- **Manual maintenance scheduling**, which can cause unnecessary downtime or delays.
- **Poor integration** with power grids and data storage systems.
- Limited **user accessibility** for monitoring system status remotely.

To address these issues, there is a need for a **smart and automated system** that can efficiently collect, analyze, and display real-time solar data while providing insights for maintenance and optimization. The proposed **Solar Panel Monitoring and Management System (SPMMS)** will enable users to monitor power output, environmental parameters (such as temperature and sunlight intensity), and system health through an interactive web and mobile interface.

Motivation:

The motivation behind developing the **SPMMS** stems from the growing importance of renewable energy and the need for smarter technologies to manage it effectively. As the world transitions toward sustainable energy, optimizing solar power usage becomes essential for both **economic** and **environmental** reasons. By creating an intelligent platform that combines **IoT sensors**, **cloud-based data analytics**, and **predictive maintenance**, we can:

- Maximize energy efficiency and cost savings.
- Reduce system downtime through early fault detection.
- Empower users with accessible, real-time insights.
- Contribute to a greener and more sustainable energy future.

The project reflects a step toward **smart energy management systems** that ensure higher reliability, better resource utilization, and long-term sustainability.

Solar Panel Monitoring and Management System (SPMMS)

2. SOFTWARE REQUIREMENT SPECIFICATION

Problem Statement

With the growing demand for renewable energy, solar power has become a primary sustainable energy source. However, managing solar panels effectively involves challenges such as real-time performance monitoring, energy efficiency tracking, maintenance prediction, and integration with power grids. A systematic software solution is required to ensure optimal energy generation, effective monitoring, and cost efficiency.

1. Introduction

1.1 Purpose

The purpose of this SRS document is to specify the requirements for the Solar Panel Monitoring and Management System (SPMMS). The software solution will help in:

- Monitoring real-time energy generation and usage.
- Tracking energy efficiency and environmental factors (sunlight intensity, temperature, voltage, current).
- Providing predictive maintenance schedules to reduce downtime.
- Integrating solar panel systems with local power grids and storage devices.
- Generating reports and analytics to support decision-making for energy providers and end-users.

The document is intended for:

- Project Managers & Stakeholders – to understand scope, budget, and schedule.
- System Developers – to implement the software based on requirements.
- Solar Technicians & Operators – to use the system effectively for management and maintenance.

1.2 Scope

The Solar Panel Monitoring and Management System (SPMMS) is a software-based solution designed for residential, commercial, and industrial solar power users. It will provide:

- Real-time monitoring of energy production, consumption, and efficiency.
- Alerts and Notifications for fault detection, energy inefficiency, and abnormal conditions.
- Predictive Maintenance using historical data and machine learning analysis.
- Integration with IoT Devices to collect sensor data from solar panels.
- Role-Based Access Control for different user categories (end-user, technician, administrator).
- Reports and Analytics for performance tracking and optimization.
- Scalability for single homes as well as large-scale solar farms.

Benefits of the system:

- Increased energy efficiency.
- Reduction of maintenance costs.
- Improved reliability of solar installations.
- Enhanced usability for end-users with dashboards and mobile apps.

1.3 Overview

The SPMMS will be built as a cloud-based application with the following layers:

- **Hardware Layer:** Solar panels, sensors, IoT communication devices.
- **Software Layer:**
 - **Backend:** Cloud servers, databases, and analytics modules.
 - **Frontend:** Web portal and mobile application.
- **User Layer:** End-users, technicians, and administrators.

The system will work on both web browsers and mobile platforms (Android and iOS).

2. General Description

2.1 Product Perspective

The SPMMS will act as an independent monitoring solution but can be integrated into existing smart grid systems. It will rely on IoT sensors to transmit data to a central server, which will process and display it through dashboards.

2.2 Product Functions

- Collect data from solar panels in real time.
- Display power generation, efficiency, and fault conditions.
- Provide energy analytics (daily, monthly, yearly reports).
- Alert users when system performance drops.
- Predict potential failures using historical data.
- Allow remote access through web and mobile apps.

2.3 User Characteristics

- **Residential Users:** Limited technical knowledge, require simple dashboards and alerts.
- **Technicians:** Skilled in handling system maintenance and require detailed data logs.
- **Administrators:** Manage user accounts, grid integration, and large-scale monitoring.

2.4 Assumptions and Dependencies

- Requires uninterrupted internet connectivity.
- Solar panels must be equipped with compatible sensors.
- Cloud infrastructure must remain reliable and scalable.

3. Functional Requirements

1. Real-Time Monitoring

- The system shall collect data (voltage, current, temperature, sunlight intensity) every 10 seconds.
- The system shall update the dashboard with no more than 2-second delay.

2. Data Management

- The system shall store at least 5 years of operational data.
- The system shall provide daily, monthly, and yearly statistics.

3. Fault Detection & Alerts

- The system shall send SMS/Email notifications in case of power drop or fault.
- The system shall trigger alarms for abnormal temperature or efficiency below threshold.

4. Predictive Maintenance

- The system shall analyze historical data to predict failures.
- The system shall provide recommended maintenance schedules.

5. User Management

- The system shall support three roles: User, Technician, Administrator.
- The system shall allow secure login and multi-user access.

6. Integration

- The system shall support integration with power grids and storage systems.
- The system shall provide API access for third-party applications.

4. Interface Requirements

4.1 User Interfaces

- **Web Dashboard:** Interactive graphs, charts, and reports.
- **Mobile App:** Simplified dashboard with notifications and quick insights.

4.2 Hardware Interfaces

- IoT-enabled devices for real-time data collection.
- Communication via Wi-Fi, ZigBee, or LoRaWAN.

4.3 Software Interfaces

- **Database:** SQL/NoSQL storage solutions.
- **Cloud:** AWS, Azure, or GCP hosting.
- **APIs:** RESTful APIs for integration with smart meters and grid systems.

5. Performance Requirements

- System shall handle up to 50,000 solar panels simultaneously.
- Dashboard update time \leq 2 seconds.
- Notification delay \leq 5 seconds.
- Availability \geq 99.9% uptime.
- Scalability to support 1 household to 100,000-panel solar farms.

6. Design Constraints

- Must comply with IEC 61724 standard for PV system monitoring.
- Must follow data encryption protocols (AES-256, SSL/TLS).

- Should run on widely available operating systems (Windows, Linux, Android, iOS).
- Mobile application must not exceed 100 MB in size.

7. Non-Functional Attributes

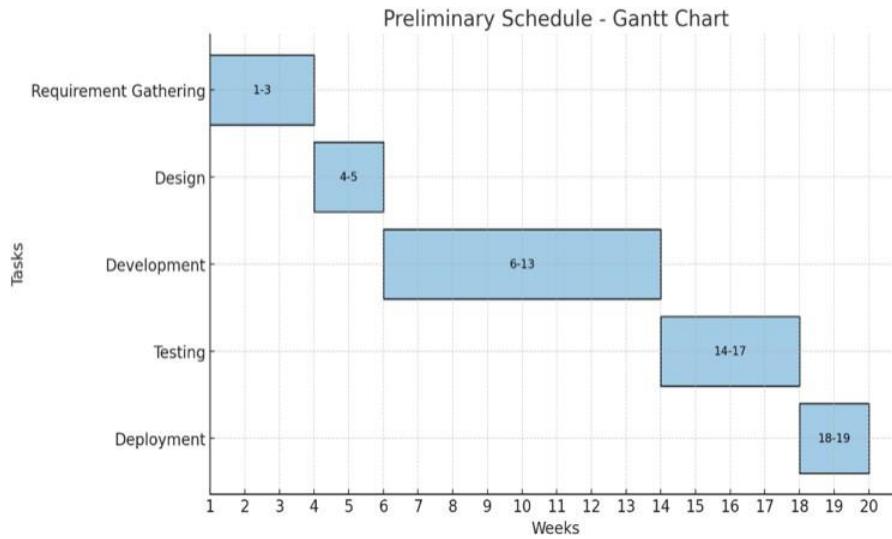
- **Reliability:** Must ensure continuous monitoring with backup servers.
- **Usability:** Simple and user-friendly dashboards.
- **Security:** Role-based authentication and two-factor login.
- **Scalability:** Must support expansion without performance degradation.
- **Maintainability:** Modular design to allow future upgrades.
- **Portability:** Web-based, Android, and iOS support.

8. Preliminary Schedule and Budget

8.1 Schedule Table

| Task | Duration(Weeks) | Weeks Covered |
|-----------------------|------------------------|----------------------|
| Requirement Gathering | 3 | Weeks 1-3 |
| Design | 2 | Weeks 4-5 |
| Development | 8 | Weeks 6-13 |
| Testing | 4 | Weeks 14-17 |
| Deployment | 2 | Weeks 18-19 |

- **Gantt Chart**



8.2 LOC Based Budget Estimation

- **Total LOC:** 32,000
- **Productivity:** 620 LOC / Person-Month
- **Labour Rate:** ₹90,000 / Person-Month
- **Cost per LOC:**

$$90,000 / 620 \approx ₹145.16$$

- **Total Cost:**

$$32,000 \times 145.16 \approx ₹46,44,960$$

- **Effort Required:**

$$32,000 / 620 \approx 52 \text{ Person-Months}$$

8.3 FP Based Budget Estimation

- **Unadjusted Function Points (UFP):** 124
- **Value Adjustment Factor (VAF):** 1.07
- **Adjusted FP (AFP):**

$$124 \times 1.07 \approx 133$$

- **LOC per FP (Average):** 50
- **Total LOC:**

$$133 \times 50 = 6650$$

- **Productivity:** 620 LOC / Person-Month
- **Labour Rate:** ₹90,000 / Person-Month
- **Cost per LOC:**

$$90,000 / 620 \approx ₹145.16$$

- **Total Cost:**

$$6650 \times 145.16 \approx ₹9,65,354$$

- **Effort Required:**

$$6650 / 620 \approx 11 \text{ Person-Months}$$

3. CLASS MODELING

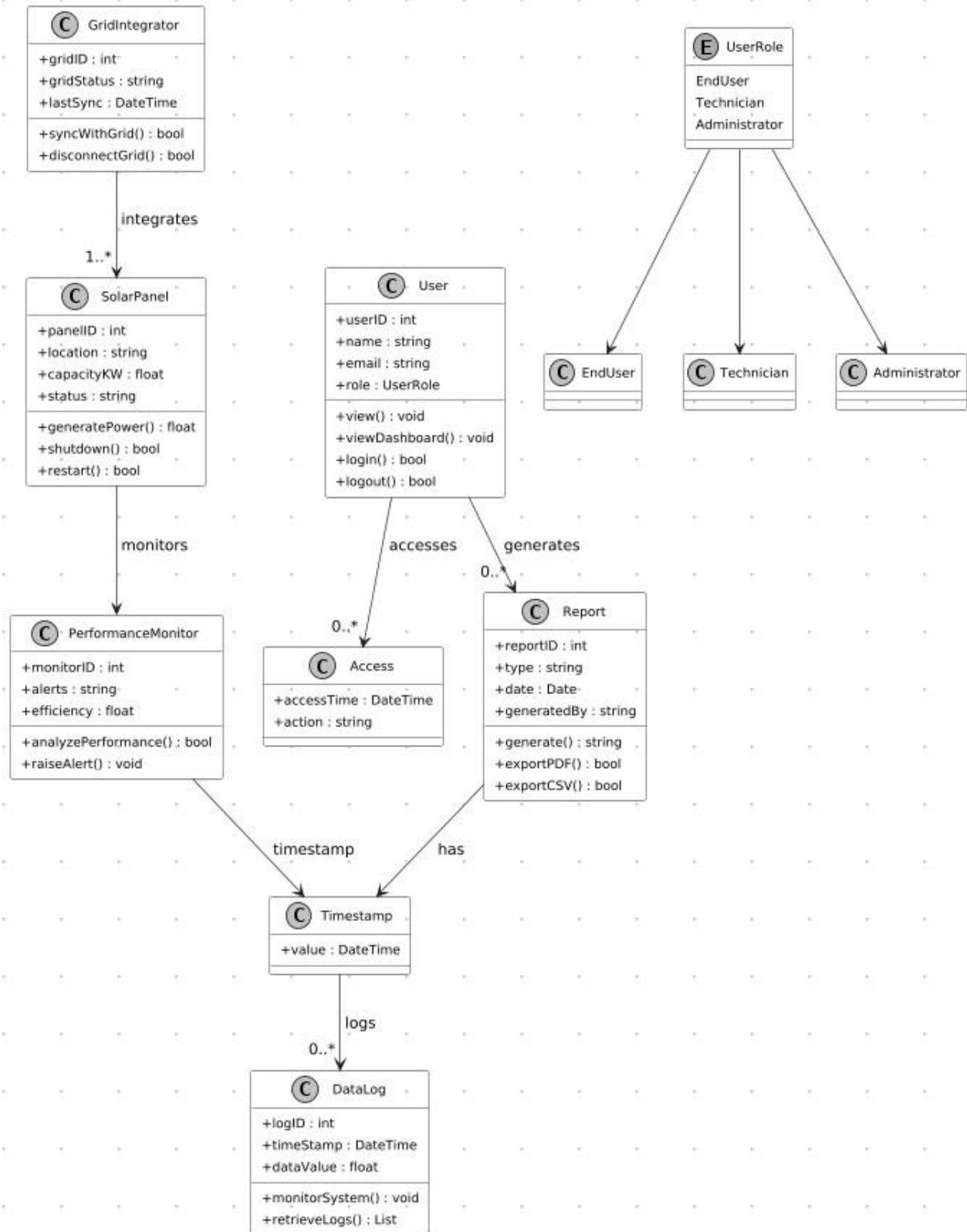


Fig 3.1

- **GridIntegrator Class:** The GridIntegrator class manages the connection between the solar panels and the main power grid. It contains attributes such as the grid ID, grid status, and the last synchronization time. Its operations include synchronizing with the grid and disconnecting from it. This class ensures that the solar power system stays connected and properly integrated with the central grid.
- **SolarPanel Class:** The SolarPanel class represents an individual solar panel in the system. It contains important attributes such as the panel ID, location, capacity in kilowatts, and the current status of the panel. The operations include generating power, shutting down, and restarting the panel. This class interacts with both the performance monitor and the grid to maintain proper functioning.
- **PerformanceMonitor Class:** The PerformanceMonitor class is responsible for analyzing the performance of solar panels. It stores attributes like the monitor ID, alerts, and efficiency. The operations in this class include analyzing performance and raising alerts when issues are detected. This class ensures the reliability and efficiency of the solar power system.
- **Timestamp Class:** The Timestamp class stores the specific DateTime value used across various components. It is used to tag logs, reports, and system events with accurate time data, supporting chronological tracking throughout the system.
- **DataLog Class:** The DataLog class stores historical monitoring data about solar panel performance. It contains attributes such as the log ID, timestamp, and data values recorded. Its operations include monitoring system values and retrieving previous logs. This class is important for reviewing performance history and generating reports.
- **User Class:** The User class stores user-related information such as the user ID, name, email, and assigned role from the UserRole enumeration. It contains operations such as viewing data, viewing the dashboard, logging in, and logging out. This class supports authentication and user interaction with the system.
- **UserRole Enumeration:** The UserRole enumeration defines the possible roles a user can have within the system. The roles include EndUser, Technician, and Administrator. These roles determine the level of access and operations available to the user.

- **EndUser, Technician, Administrator Classes:** These classes extend from the User class. EndUser is meant for general usage and basic system interaction. Technician has additional authority for monitoring and handling system alerts. Administrator holds the highest level of access, managing system configurations and user accounts. These subclasses support role-based access control in the system.
- **Access Class:** The Access class records user activities within the system. It contains attributes such as the timestamp of access and the action performed. This class is used for tracking user behavior, auditing, and ensuring system security.
- **ReportID Class:** The ReportID class stores the unique identifier for generated reports. It contains a single attribute representing the report value. This class helps associate reports with the correct log and performance data.
- **Report Class:** The Report class represents reports generated within the system. It contains attributes such as report ID, type, date, and the user who generated the report. Its operations include generating reports, exporting them as PDF, and exporting them as CSV. This class provides a structured way to review system performance and activities

4. STATE MODELING

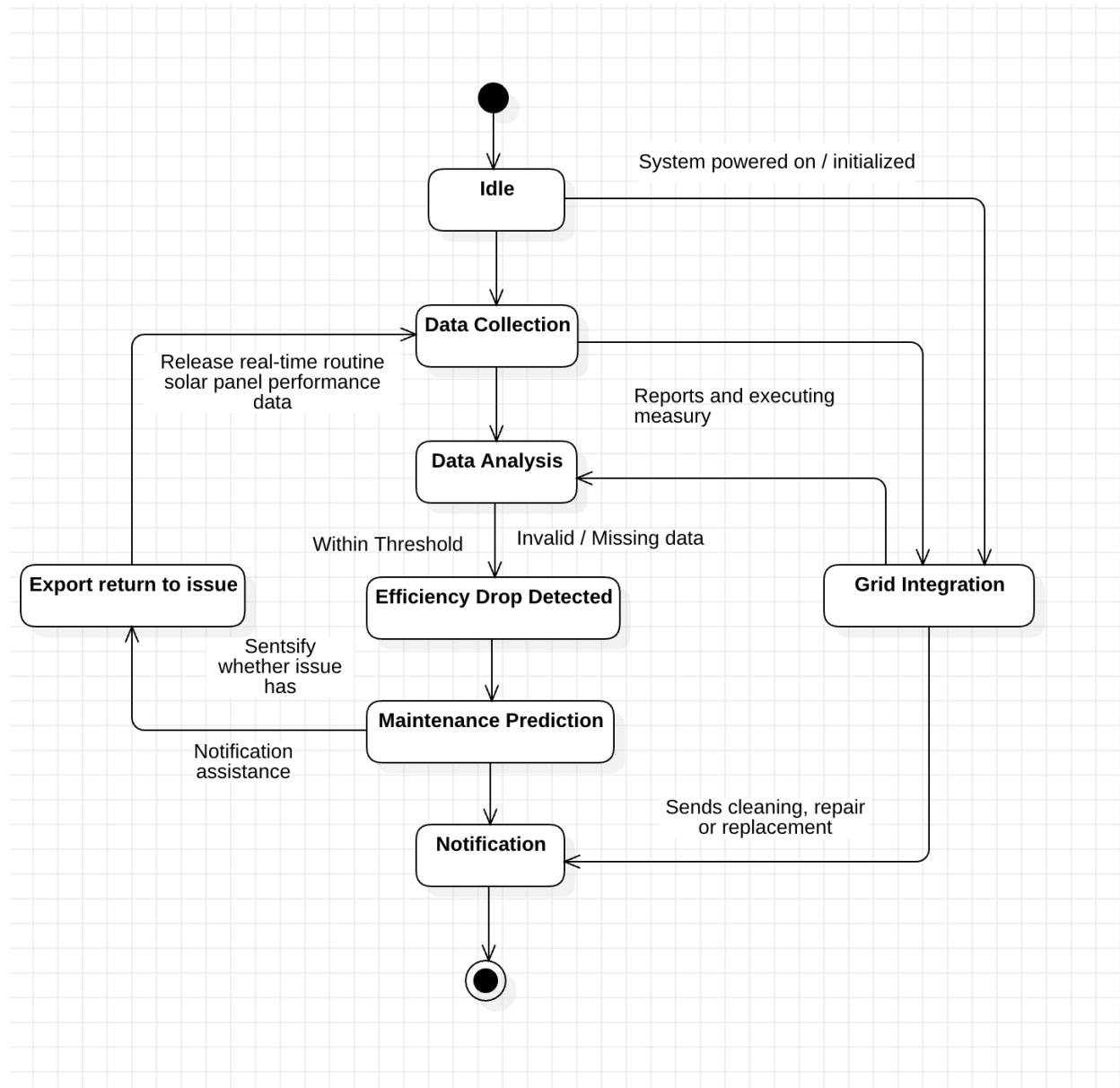


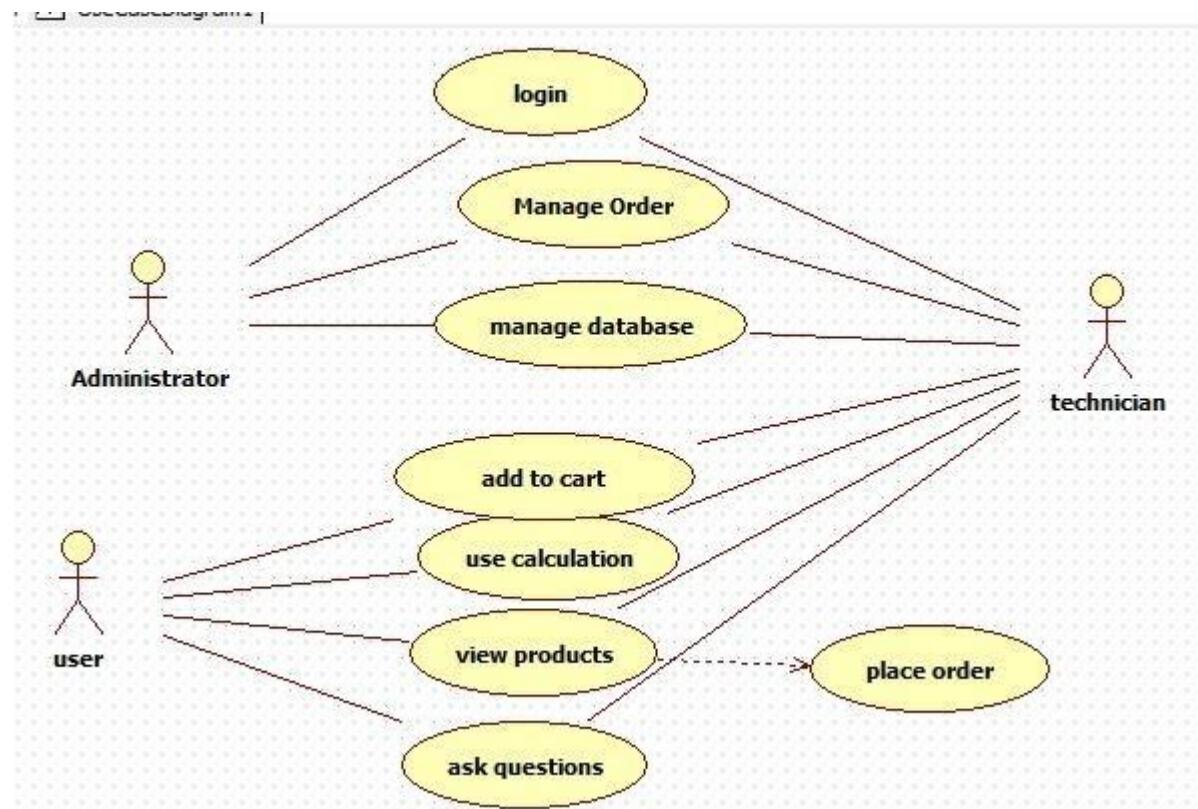
Fig 4.1

- **Idle:** This is the initial state of the system. When the system is powered on or initialized, it remains idle until solar panel data begins to flow. No analysis or reporting is performed in this state.
- **Data Collection:** Once active, the system transitions to the Data Collection state. Here, real-time solar panel performance data—such as voltage, current, temperature, and energy output—is gathered from sensors or devices. This data is forwarded for processing.

- **Data Analysis:** After collecting data, the system enters the Data Analysis state. The data is validated and analyzed to determine overall performance. If the data is missing, invalid, or inconsistent, corrective measures are triggered. Valid reports are generated and forwarded to the next states.
- **Efficiency Drop Detected:** If the analyzed performance falls below the expected threshold, the system transitions to the Efficiency Drop Detected state. This state identifies issues such as reduced energy production, panel overheating, shading problems, or component malfunction.
- **Maintenance Prediction:** Based on the detected efficiency drop, the system predicts the type of maintenance required. This can include cleaning, repair, calibration, or replacement. The system evaluates issue severity and prepares proper maintenance suggestions.
- **Notification:** Once maintenance needs are identified, the system moves to the Notification state. Here, the user receives alerts about issues, predicted failures, recommended actions, or urgent warnings through the dashboard, email, or SMS.
- **Export Return to Issue:** If the user or technician requests a detailed report, the system enters this state. It exports issue summaries, performance logs, or historical data to help diagnose the problem more effectively.
- **Grid Integration:** During all stages, the system may transition to Grid Integration. In this state, SPMS coordinates with the power grid to optimize energy distribution, adjust load, and ensure safe grid-connected operations.

5. INTERACTION MODELING

5.1 Use Case Diagram



5.1.1 Actors

1. **Administrator:** A stick-figure labeled “Administrator” — this role has permissions to do high-level system operations (e.g., manage orders and database).
2. **Technician:** Another stick-figure labeled “technician” — this role interacts with various system functions like order management and database management.
3. **User:** A third stick-figure labeled “user” — this is the regular/customer user of the system who interacts mostly with the product catalog and ordering.

5.1.2 Use Cases

These are represented by ovals (ellipses), each naming a distinct function or service the system offers:

4. **Login:** Actors (Administrator and Technician) both need to authenticate into the system.

5. **Manage Order:** Admin and Technician can manage orders — presumably create, update, process orders.
6. **Manage Database:** Admin and Technician can manipulate or maintain the system's database (e.g., product data, user data).
7. **Add to Cart:** The User can add products to their shopping cart.
8. **Use Calculation:** The User can perform some calculation (maybe cost, shipping, or price estimation).
9. **View Products:** The User can browse or view the catalog of products.
10. **Ask Questions:** The User can ask questions (probably about products, orders, or support).
11. **Place Order:** This is depicted slightly separately (off to the side) and is dashed-lined to “View Products,” suggesting a dependency or relationship (see below).

5.1.3 Relationships

12. **Associations:** Solid lines connect actors to the use cases they participate in, showing who can do what. For example, User → Add to Cart, User → View Products, Administrator → Manage Database, etc.
13. **Dependency / Use Relationship:** The dashed arrow from “View Products” to “Place Order” indicates that “Place Order” depends on or is related to “View Products.” It could mean that before placing an order, the user typically views the products. The diagram doesn’t explicitly label it as `«include»` or `«extend»`, but visually it is like a dependency.

5.1.4 System Boundary (Implicit)

Although not explicitly drawn as a box, we can infer that the ovals (use cases) represent the functions inside the system’s boundary. The actors lie outside that boundary, interacting with the system.

5.1.5 Interpretation & Purpose in Context

- **High-Level Overview of System Functionality:** This use-case diagram gives a bird’s-eye view of what functionality the system offers and who can interact with each part. Use-case diagrams are especially useful to communicate with stakeholders (both technical and non-technical) what the system *will do*.
- **Identifying Roles and Permissions:** By mapping out which actor is connected to which

use case, this diagram clarifies role-based access — for instance, only Admin and Technician handle system-level management (orders, database), while the regular User interacts with product browsing and ordering.

- **Defining Functional Requirements:** The use cases shown are the key functional requirements for the system. Each oval corresponds to something the system must support. According to UML theory, use case diagrams help capture these requirements from the user's perspective.
- **Scope Clarification:** The diagram delineates what is part of the system (use cases) versus external interactions (actors), helping define the system boundary.
- **Modularity & Maintenance:** Some use cases might share common functionality or could be modularized. For example, if “Place Order” is always preceded by viewing products, or if “Login” is required for other operations, this relationships can help in designing the system more cleanly. Use-case relationships like *include* or *extend* support this.

5.2 SEQUENCE DIAGRAM

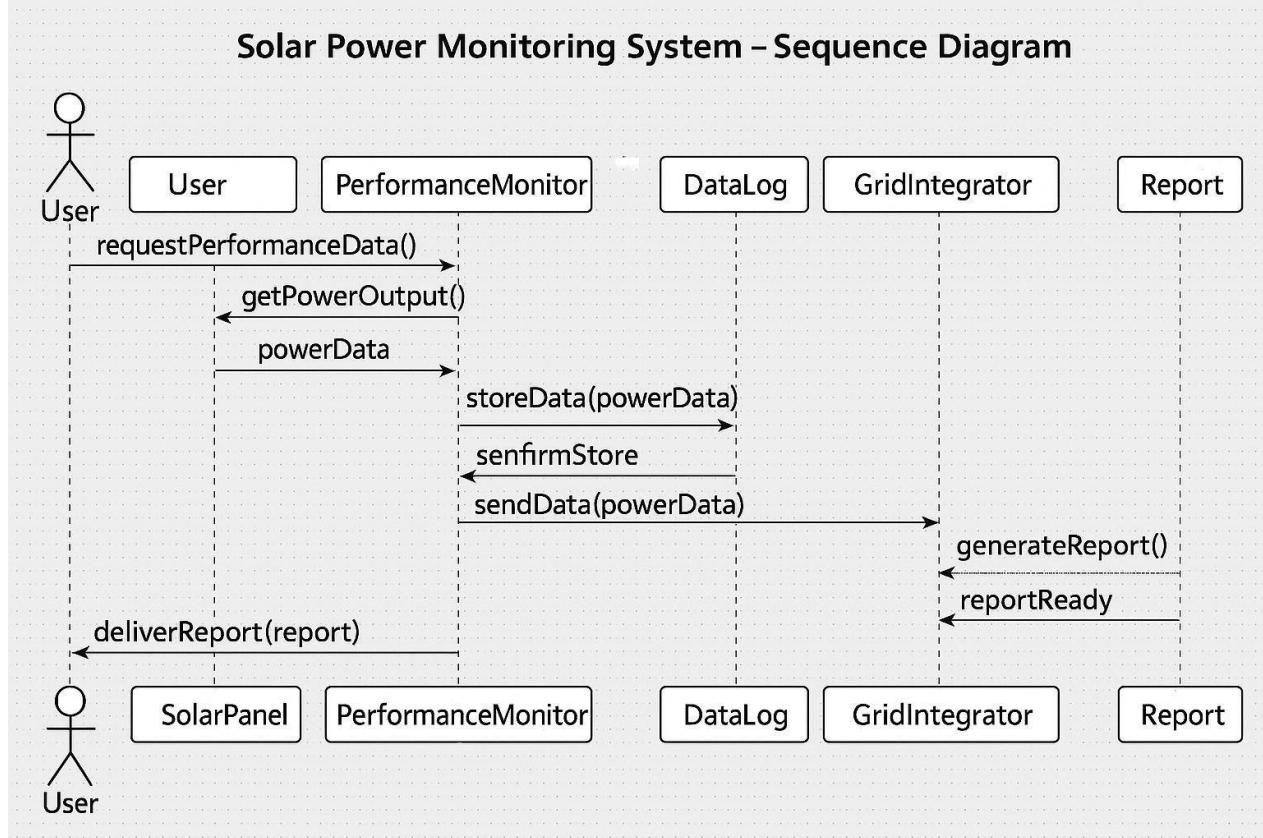


Fig 5.2

Lifelines

- **User**

This lifeline represents the person interacting with the solar monitoring system. The user initiates the sequence by requesting performance data and receives the final performance report generated by the system.

- **PerformanceMonitor**

The PerformanceMonitor lifeline represents the component responsible for collecting and processing solar panel performance data. It receives requests from the user, interacts with the SolarPanel to obtain power readings, and coordinates with DataLog and GridIntegrator.

- **SolarPanel**

This lifeline represents the solar panel hardware or sensor module that provides actual power

output data. It responds to PerformanceMonitor with real-time powerData when requested.

- **DataLog**

The DataLog lifeline is responsible for storing the collected solar performance data. It receives powerData from the PerformanceMonitor, stores it, and sends a confirmation back.

- **GridIntegrator**

The GridIntegrator lifeline represents the system that integrates processed solar data with the grid analysis module. It receives stored power data, generates analysis or reports when requested, and sends back the report status.

- **Report**

The Report lifeline represents the entity responsible for generating the final performance report. It is triggered by GridIntegrator and returns the completed report.

Interactions

- **User → PerformanceMonitor: requestPerformanceData()**

The user sends a request to view solar performance data.

- **PerformanceMonitor → SolarPanel: getPowerOutput()**

PerformanceMonitor requests real-time power output from the solar panel.

- **SolarPanel → PerformanceMonitor: powerData**

The solar panel replies with the current power output data.

- **PerformanceMonitor → DataLog: storeData(powerData)**

The collected data is sent to the DataLog to be stored.

- **DataLog → Performance Monitor: confirmStore**

DataLog confirms that the data has been successfully saved.

- **PerformanceMonitor → GridIntegrator: sendData(powerData)**

PerformanceMonitor forwards the stored solar data to the GridIntegrator for grid-level evaluation.

- **GridIntegrator → Report: generateReport()**

GridIntegrator requests the report module to generate a detailed performance report.

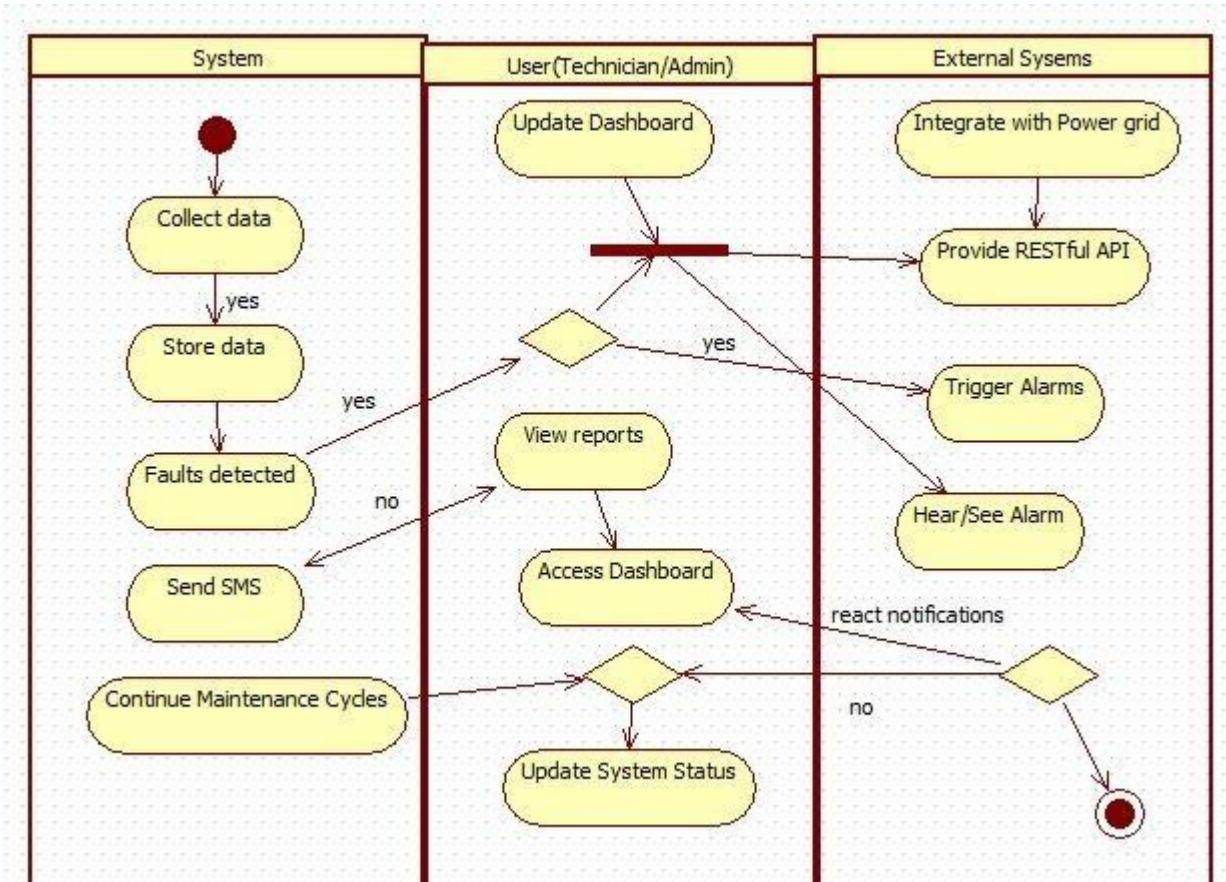
- **Report → GridIntegrator: reportReady**

The report module completes and sends back the generated report.

- **PerformanceMonitor → User: deliverReport(report)**

Finally, the PerformanceMonitor sends the completed performance report to the user.

5.3 Activity Diagram



The diagram represents the **end-to-end workflow of a monitoring and maintenance system**, involving three major components:

1. **System** (automated internal processes)
2. **User (Technician/Admin)**
3. **External Systems** (power grid integration, alarm system, APIs)

The activity diagram shows how data is collected, processed, and acted upon, and how users and external systems interact with these automated workflows.

5.3.1. System Swimlane

a. Collect Data

The system begins by automatically collecting operational data from sensors, devices, or field units.

b. Store Data (Decision: yes)

If data is successfully collected, the system proceeds to store the data in a database or internal memory.

c. Faults Detected? (Decision Node)

The system checks whether the collected data indicates abnormal conditions, errors, or faults.

- **Yes → Fault Detected:**

The system triggers the next step: sending an SMS alert.

- **No → Normal Condition:**

The system skips the alert and goes directly to continuing maintenance cycles.

d. Send SMS

If a fault is detected, the system sends an SMS notification to concerned personnel (e.g., technicians/admins).

e. Continue Maintenance Cycles

Regardless of faults, the system loops back to its regular monitoring and maintenance cycle, maintaining continuous operation.

5.3.2. User (Technician/Admin) Swimlane

a. Update Dashboard

Users access the system dashboard to modify, view, and monitor real-time system performance.

This step links to multiple external and internal actions.

b. Decision: Authenticate/Validate?

A decision node checks whether the dashboard update request is valid or whether user action is required.

c. View Reports (If yes)

If validation is successful, the user can view system-generated reports, trends, fault logs, and performance summaries.

d. Access Dashboard

Users proceed to navigate into the main dashboard interface. This step branches into two possible directions:

e. Decision: Modify System Status?

The user may choose to update or not update system status.

- **Yes → Update System Status**
The user modifies parameters such as operational settings, thresholds, maintenance indicators, alarms, etc.
- **No → End of User Activity Flow**
The user exits the workflow.

5.3.3. External Systems Swimlane

a. Integrate with Power Grid

The system interfaces with the electricity grid for real-time data sharing, load balancing, or grid-aware control.

b. Provide RESTful API

External systems expose APIs that interact with the monitoring system's backend—enabling automation, data transfer, remote commands, and third-party integration.

c. Trigger Alarms

If external conditions or system communications indicate an issue, the external alarm unit is triggered.

d. Hear/See Alarm

Technicians or nearby personnel receive visual or audible alarms.

e. React to Notifications (Decision Node)

A decision checks whether the user reacts to the alarm notifications:

- **Yes → System/User continues workflow**
The user or system acknowledges the alarm and takes appropriate action (redirects into dashboard update flow).

- **No → External Flow Ends**
If no response is given, the external activity terminates at an end node.

5.3.4 Overall Workflow Summary

This activity diagram illustrates the **complete functional behavior** of a monitoring and maintenance platform:

System Side:

- Automatically collects and stores data
- Detects failures
- Sends SMS alerts
- Runs continuous monitoring cycles

User Side:

- Views dashboards and reports
- Updates system status
- Responds to notifications and alarms

External Systems:

- Interface with the power grid
- Provide APIs for integration
- Trigger alarms based on critical conditions
- Notify users through audible/visual alerts