Mini Meter Reading and Billing System

Software Engineering Department Faculty of Information Technology - Sabratha University

Course: Software Design Patterns ITSE424 - Semester Fall 2025

Submitted on 20 Dec 2025 by Ayoub Ahfeeth Aboulqasim Abu Isnaynah & Nojaid Abdullah Issa Aummara

To Dr. Mai Muftah Elbaabaa.

Abstract –This project presents the design and implementation of a Mini Meter Reading and Billing System, developed as part of the *Software Design Patterns* course. The system simulates a smart electricity meter environment where users can register, submit meter readings, generate electricity bills, make payments, recharge balances, and submit service reports. The application is implemented using Java and follows the Model–View–Controller (MVC) architectural pattern to separate presentation, business logic, and data management concerns. A DAO (Data Access Object) layer is used to manage database operations and ensure loose coupling with the persistence layer. Several well-known software design patterns are applied to improve modularity, flexibility, and maintainability of the system. The system relies on a relational database to store users, readings, bills, payments, logs, and reports, providing a structured and reliable billing solution.

Keywords: Software Design Patterns, MVC, Java, Billing System, Database Design.

1. Introduction

Electricity billing systems play an important role in managing energy consumption and payments. With the advancement of smart systems, it has become necessary to design applications that are flexible, maintainable, and scalable.

This project implements a Mini Meter Reading and Billing System that allows users to interact with a simulated smart meter environment. The system supports user registration and authentication, meter reading submission, bill generation, payment processing, balance recharge, and report submission, the main objective of this project is not only to implement a functional billing system but also to demonstrate the practical application of software design patterns learned during the course. By applying multiple design patterns from different categories, the system achieves better separation of concerns, easier maintenance, and improved extensibility.

The system supports two main roles: **User** and **Admin**, users interact with meter readings and billing operations, while Admin users are responsible for monitoring system data, managing users, and reviewing reports.

2. Objectives

The main objectives of this project are as follows:

- To design and implement a mini meter reading and billing system using Java.
- To apply the Model–View–Controller (MVC) architectural pattern for clear separation of concerns.
- To demonstrate the practical use of software design patterns such as Singleton, Observer, Factory, Strategy, Template Method, and Facade.
- To design a relational database that ensures data integrity and supports billing and transaction operations.
- To improve system maintainability, scalability, and code organization through proper architectural design.

3. System Architecture

The system follows the Model–View–Controller (MVC) architecture combined with a layered design.

3.1 MVC Overview

- **Model:**
  Represents the core data of the system such as Admin, User, Bill, MeterReading, Payment, Report, and Log.
- **View:**
  Handles user interaction through console-based menus and displays system output such as balances, bills, and messages.
- **Controller:**
  Acts as an intermediary between the View and the Service layer. It receives user input and coordinates system operations.

3.2 Layered Architecture

- **Controller-Layer:**
  Manages user requests and navigation (e.g., UserController, AdminController).
- **Service-Layer:**
  Contains business logic such as billing, payments, and validation (UserServiceImpl, BillingServiceImpl, AdminServiceImpl).
- **DAO-Layer:**
  Handles database operations and SQL queries.
- **Database-Layer:**
  Stores system data including users, readings, bills, payments, logs, and reports.

This architectural approach improves modularity and maintainability by clearly separating user interface logic from business rules and data access operations, it also allows individual components to be modified or extended independently without affecting the overall system

behavior.

As illustrated in **Figure 1**, the MVC-based layered architecture ensures a clear separation of concerns, where controllers manage user requests, services handle business logic, and DAOs are responsible for data persistence.

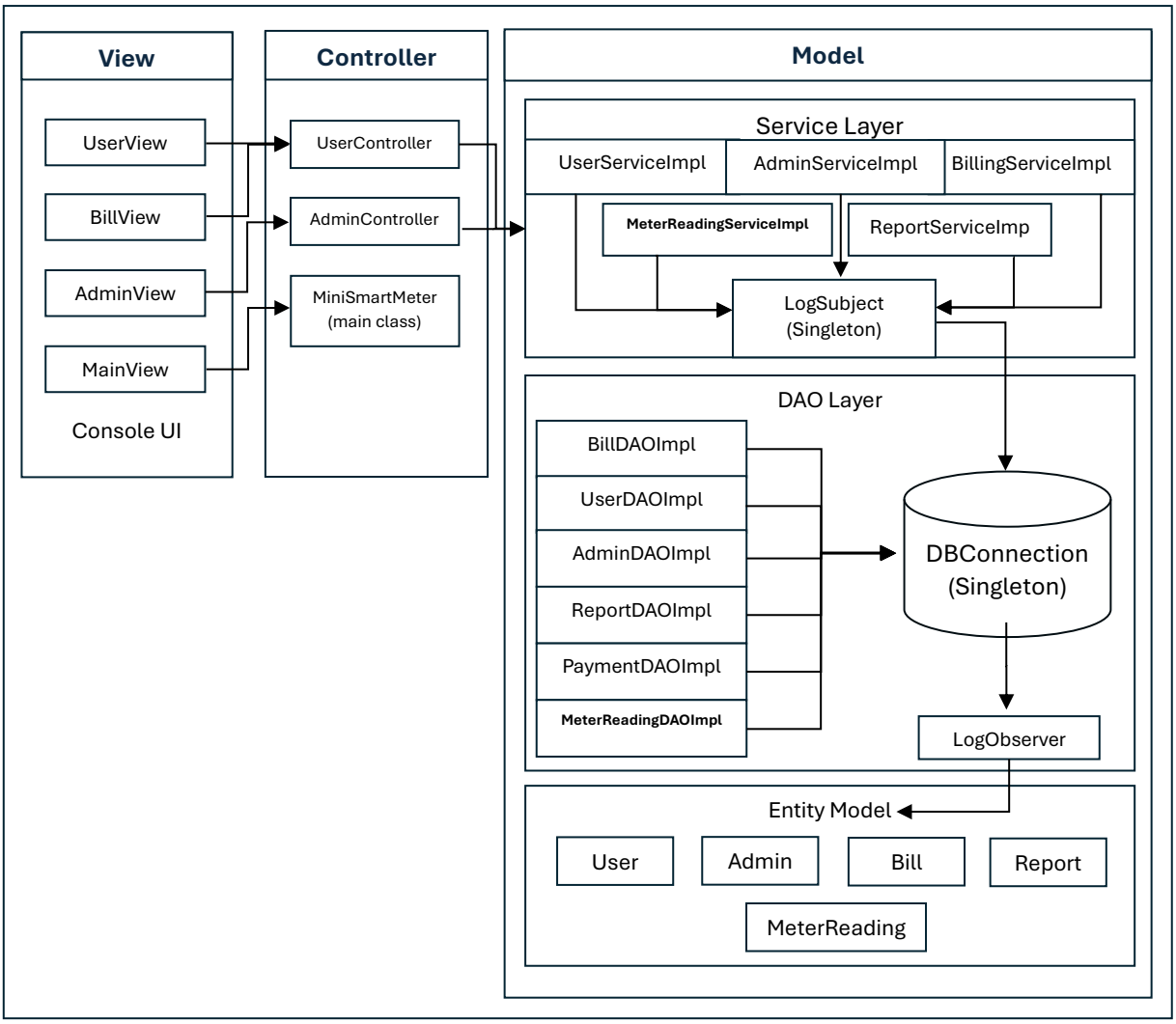

Figure 1: MVC and Layered Architecture of the Mini Meter Reading

4. Design Patterns Applied

This project applies several design patterns from different categories as required by the course.

4.1 Singleton Pattern (Creational)

Usage:

- Database connection (DBConnection)
- Centralized logging subject (LogSubject)

Why Singleton?

The Singleton pattern ensures that only one instance of the database connection and logging subject exists throughout the application. This prevents resource waste, avoids inconsistent states, and provides a single point of access.

4.2 Observer Pattern (Behavioral)

Usage:

- Logging system activities such as login, balance recharge, bill generation, and report submission.

Implementation:

- LogSubject maintains a list of observers.
- LogObserver listens for events and saves logs to the database.

Why Observer?

It decouples logging logic from business logic and allows new observers to be added without modifying existing code.

4.3 Factory Pattern (Creational)

Usage:

- Payment processing (Visa, PayPal, LibiPay, MobiCash).

Why Factory?

The Factory pattern encapsulates the creation of payment methods and allows new payment options to be added easily without changing client code.

4.4 Strategy Pattern (Behavioral)

Usage:

- Billing calculation (Normal, Peak, Weekend).

Why Strategy?

Different billing rules are encapsulated into separate strategies, allowing billing behavior to change dynamically at runtime.

4.5 Template Method Pattern (Behavioral)

Usage:

- Bill generation workflow.

Why Template Method?

It defines a fixed sequence for generating bills while allowing subclasses to customize specific calculation steps.

4.6 Facade Pattern (Structural)

Usage:

- BillingFacade.

Why Facade?

It simplifies complex interactions between billing services, DAOs, and models, providing a clean and simple interface for controllers.

5. System Workflow

1. The user registers or logs in.
2. The user submits meter readings.
3. The admin logs in.
4. The admin views all users.
5. The admin views all reports.
6. The system calculates consumption.
7. The user selects the billing type.
8. The bill is generated and stored as txt file.
9. The user pays the bill using a selected payment method.
10. The system updates the balance and records the payment.
11. Logs are automatically saved using the Observer pattern.

6. Database Design (ERD)

The database includes the following main entities: User, Admin, MeterReading, Bill, Payment, Report, and Log, the relationships between entities are defined using one-to-many (1:M) and one-to-one (1:1) associations based on system requirements, foreign keys are used to link related tables and enforce referential integrity between records, this design ensures data consistency, minimizes redundancy, and supports reliable storage and retrieval of billing and transaction data.

Additionally, one-to-many (1:M) relationships are used to represent interactions such as users submitting multiple meter readings and making multiple payments. One-to-one (1:1)

relationships are applied where unique associations are required, such as linking a specific bill to a specific meter reading. This structured ERD design enhances database clarity, supports scalability, and ensures accurate representation of real-world billing processes.

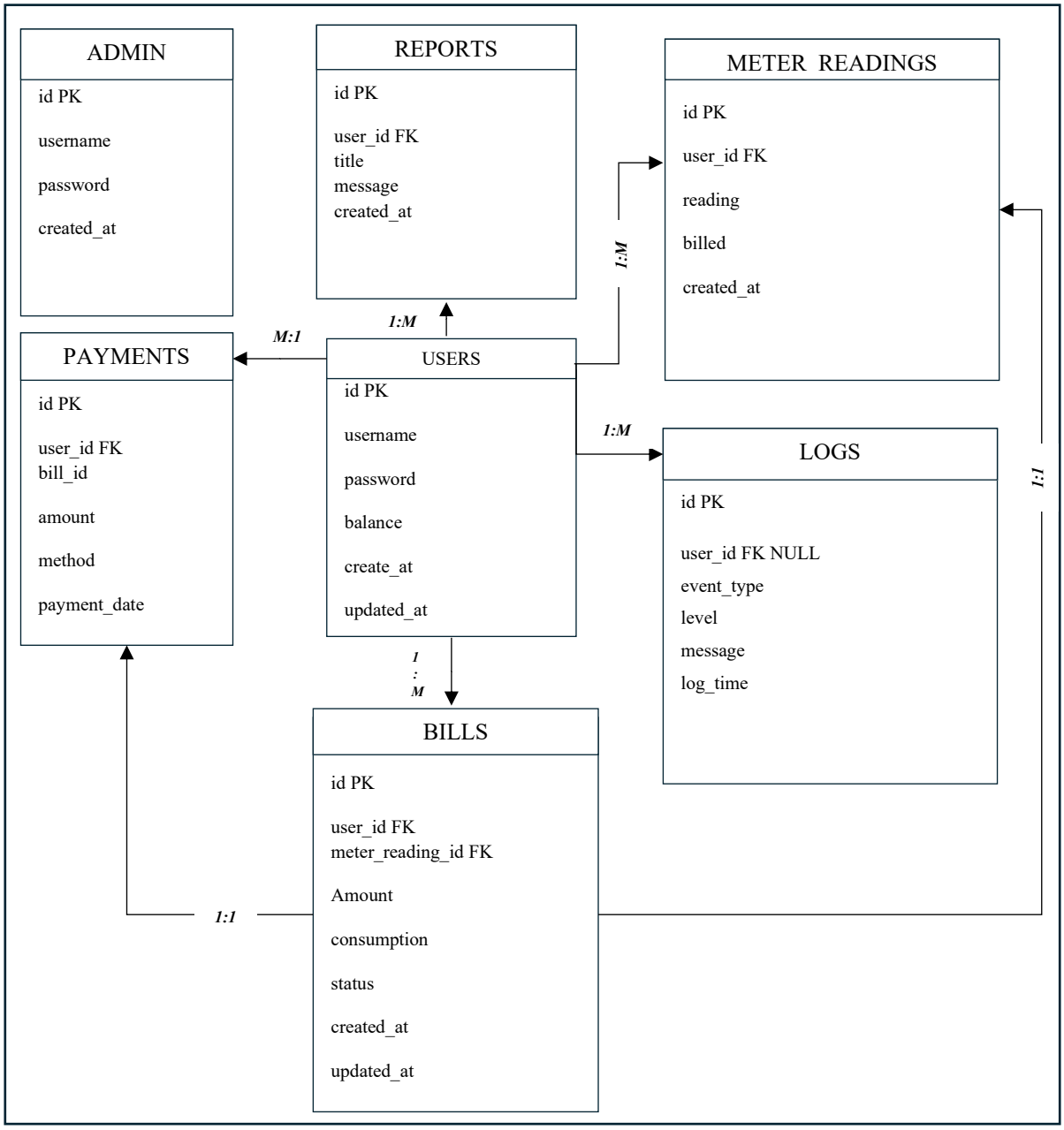

Figure 2: Database Entity–Relationship Diagram (ERD) of the Mini Meter Reading and Billing System

7. Conclusion

This project demonstrates the effective use of multiple software design patterns in a real-world Java application. By combining MVC architecture with well-known design patterns, the system achieves high modularity, low coupling, and improved maintainability. The Mini Meter Reading and Billing System serves as a practical example of how design patterns can be applied to solve common software design problems in a structured and professional manner.

8. References

[1] Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994).
Design Patterns: Elements of Reusable Object-Oriented Software.
Addison-Wesley.

[2] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996).
Pattern-Oriented Software Architecture, Volume 1: A System of Patterns.
Wiley.

[3] Oracle.
Java Design Patterns Documentation.

[4] Oracle. Model-View-Controller (MVC) Architecture.