



## **BUSINESS COMPONENT DEVELOPMENT - I (CMP6222)**

**Student First Name : ABDUL RAHUMAN**

**Student Last Name : MUHAMMED HAZEEM**

**BCU No. : 22178960**

**NIC No. :199835410549**



## Contents

1. Introduction.....	3
2. Situation.....	3
3. Solution.....	3
4. Architectural Diagram.....	5
5. Sequence Diagram.....	6
6. Source Code.....	7
7. Graphical User Interface.....	12
8. Efficiency and Correctness of the solution.....	13
9. Reference.....	14

## **1. Introduction**

The increasing demand for energy worldwide has led to the development of various technologies to optimize energy consumption and reduce waste. Smart Energy Management System (SEMS) uses advanced sensors and analytics to monitor and control energy usage in homes, buildings, and industries. The SEMS typically includes an Internet of Things (IoT) device that captures and transmits sensor data such as voltage, frequency, and current to a messaging server, which in turn processes the data with an analytical server to monitor and optimize energy usage.

In this report, we present a research study on the implementation of an IoT device in the SEMS to capture and transmit sensor data, and the development of a Java SE application to simulate the IoT device input. The objective of this research is to evaluate the performance and effectiveness of the SEMS in monitoring energy usage and optimizing energy consumption, using the data collected by the IoT device and processed by the analytical server. Specifically, we aim to monitor the total power consumption, average current, average voltage, and average frequency from the sensor data, and assess the impact of the SEMS on energy efficiency and cost savings.

## **2. Situation**

The Smart Energy Management System is going to implement an IoT device to capture and transmit its sensors' data (voltage, frequency and current) to the messaging server which tries to process data with an analytical server, where their requirement is to monitor the (total power consumption, average current, average voltage, and average frequency) from that data.

## **3. Solution**

In Smart Energy Management System, IoT devices are capturing the sensor's data such as voltage, frequency, current and sending the help of messaging server to the EJB Module. Business logics are written in the EJB module. EJB module is connected with database and storing the sensor's data. Users can view the analytical interface in a web application.

To solve the problem of the Smart Energy Management System (SEMS) with an IoT device, we can use Enterprise JavaBeans (EJB) components as part of the business layer of the J2EE platform solution. The EJB components provide a scalable, distributed, and transactional architecture for handling the processing and analysis of the sensor data from the IoT device.

Here is an explanation of the EJB components used in the solution:

Session beans:

- Used to encapsulate the business logic of the SEMS, such as data analytics and report generation.
- Can be stateless or stateful, depending on the need for maintaining session context and data between requests from the presentation layer.
- Can be accessed by remote clients using Remote Method Invocation (RMI) or web services.

Message-driven beans:

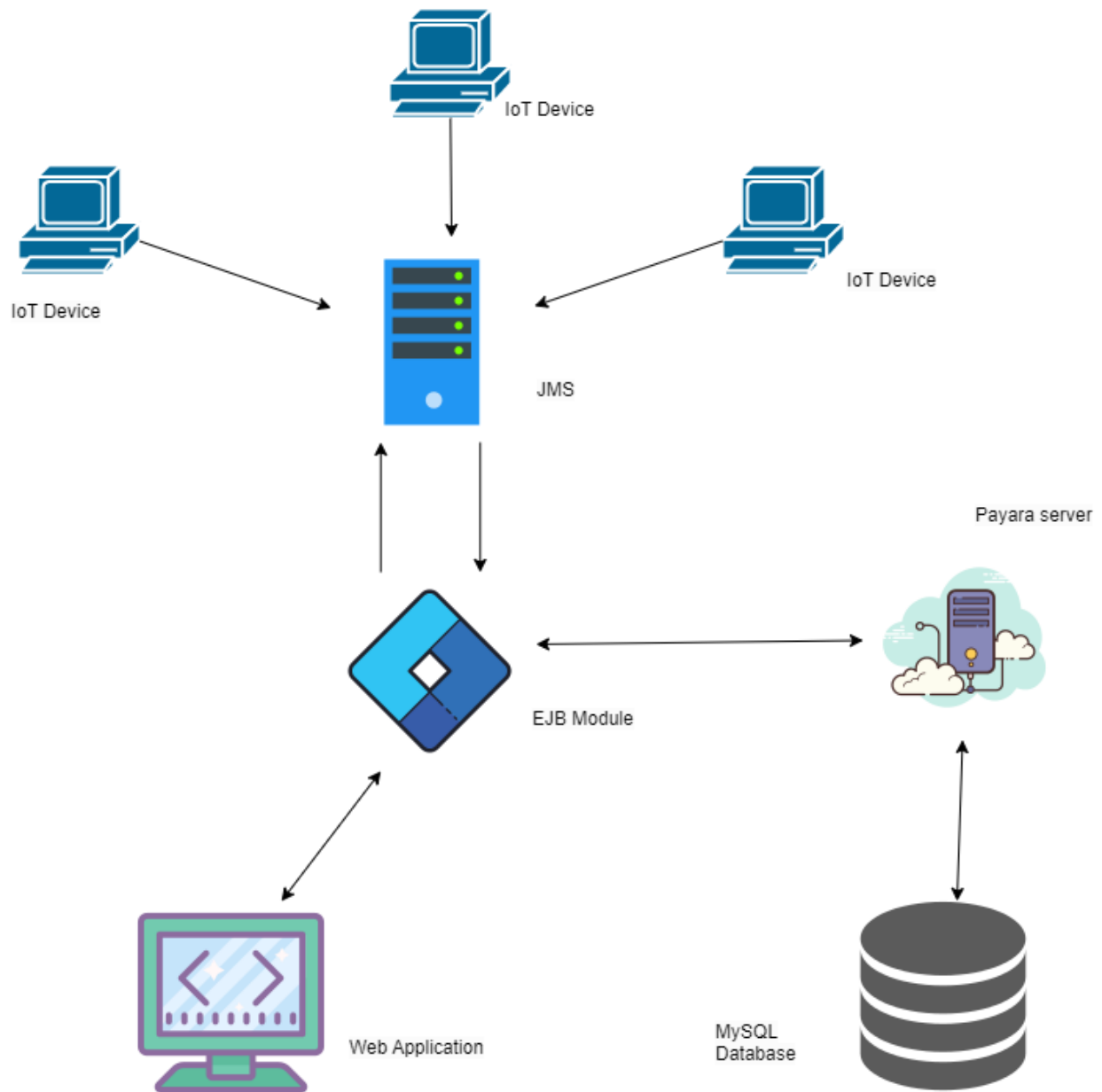
- Used to handle the asynchronous communication and messaging between the IoT device and the SEMS, using the Java Message Service (JMS) API.
- Receive and process messages from the IoT device, such as sensor data and status updates, and perform the necessary actions or trigger events in the SEMS.

Entity beans:

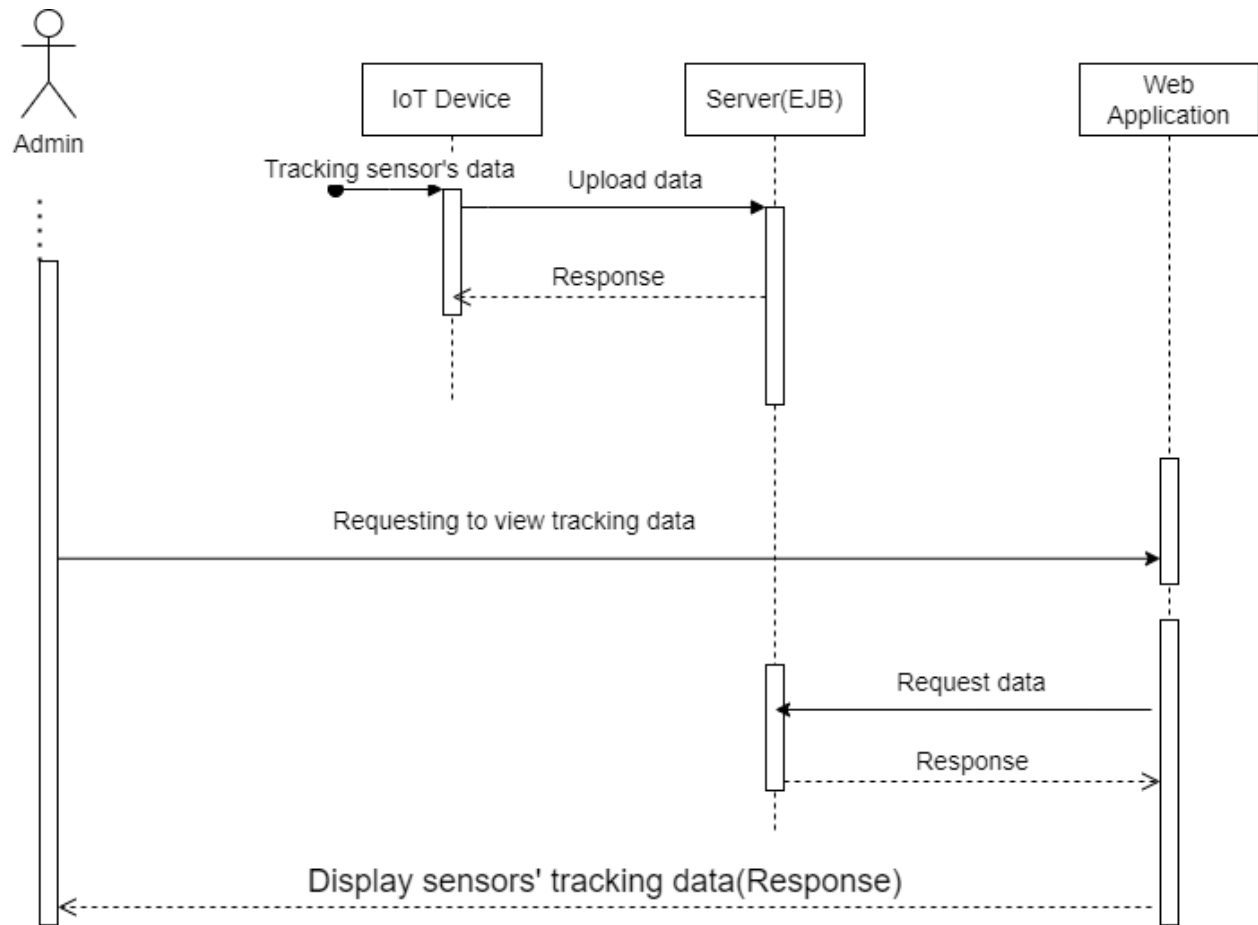
- Used to represent and manage the persistence of the sensor data and analytical results in the relational database.
- Can be mapped to the database schema using the Java Persistence API (JPA) or the Container-Managed Persistence (CMP) mechanism of EJB.
- Provide transactional consistency and concurrency control for data access and modification.

Overall, the use of EJB components in the SEMS solution provides a robust and scalable architecture for handling the complex business logic and data processing required by the IoT device and the analytical server. By using stateless or stateful session beans, message-driven beans, entity beans, and timer service, we can ensure the efficient and reliable communication, processing, and storage of sensor data and analytical results, while providing a high level of transactional consistency and reliability.

#### 4. Architectural Diagram



## 5. Sequence diagram



## 6. Source Code

### EJBLibrary

#### AnalyticalServer.java

```
package com.sems.modul;

import javax.ejb.Remote;

@Remote

public interface AnalyticalServer {

    void processSensorData();

    double getTotalPowerConsumption();

    double getAverageCurrent();

    double getAverageVoltage();

    double getAverageFrequency();

}
```

### EJBLibrary

#### MessagingServer.java

```
package com.sems.modul;

import javax.ejb.Remote;

@Remote

public interface MessagingServer {

    void receiveData(String voltage, String frequency, String current);

}
```

## **EJBModule**

### **AnalyticalServerImpl.java**

```
package com.sems.modul;
```

```
import java.util.ArrayList;
```

```
import javax.ejb.Stateful;
```

```
import javax.ejb.Stateless;
```

```
@Stateful
```

```
public class AnalyticalServerImpl implements AnalyticalServer {
```

```
    private double totalPowerConsumption;
```

```
    private double totalCurrent;
```

```
    private double totalVoltage;
```

```
    private double totalFrequency;
```

```
    private int count;
```

```
    MessagingServerImpl messagingServer = new MessagingServerImpl();
```

```
    @Override
```

```
    public void processSensorData() {
```

```
    }
```

```
    @Override
```

```
    public double getTotalPowerConsumption() {
```



```
ArrayList<String> voltage = messagingServer.voltage;
ArrayList<String> current = messagingServer.current;
double vTotal = 0;
for (String v : voltage) {
    vTotal += Double.parseDouble(v);
}

double cTotal = 0;
for (String c : current) {
    cTotal += Double.parseDouble(c);
}

double tpc = vTotal * cTotal * 0.8;

return totalPowerConsumption;
}
```

@Override

```
public double getAverageCurrent() {
    ArrayList<String> current = messagingServer.current;
    double cTotal = 0;
    double count = 0;
    for (String c : current) {
        cTotal += Double.parseDouble(c);
        count += 1;
    }

    return cTotal/ count;
}
```

```
}
```

```
@Override
```

```
public double getAverageVoltage() {  
    ArrayList<String> voltage = messagingServer.voltage;  
    double cTotal = 0;  
    double count = 0;  
    for (String c : voltage) {  
        cTotal += Double.parseDouble(c);  
        count += 1;  
    }  
  
    return cTotal/ count;  
}
```

```
@Override
```

```
public double getAverageFrequency() {  
  
    ArrayList<String> frequency = messagingServer.frequency;  
    double cTotal = 0;  
    double count = 0;  
    for (String c : frequency) {  
        cTotal += Double.parseDouble(c);  
        count += 1;  
    }  
  
    return cTotal/ count;  
}
```

```
}
```

## **EJBModule**

MessagingServerImpl.java

```
package com.sems.modul;
```

```
import java.util.ArrayList;
```

```
import javax.ejb.Stateful;
```

```
import javax.ejb.Stateless;
```

```
@Stateful
```

```
public class MessagingServerImpl implements MessagingServer {
```

```
    public ArrayList<String> voltage = new ArrayList<>();
```

```
    public ArrayList<String> frequency = new ArrayList<>();
```

```
    public ArrayList<String> current = new ArrayList<>();
```

```
    @Override
```

```
    public void receiveData(String voltage, String frequency, String current) {
```

```
        this.voltage.add(voltage);
```

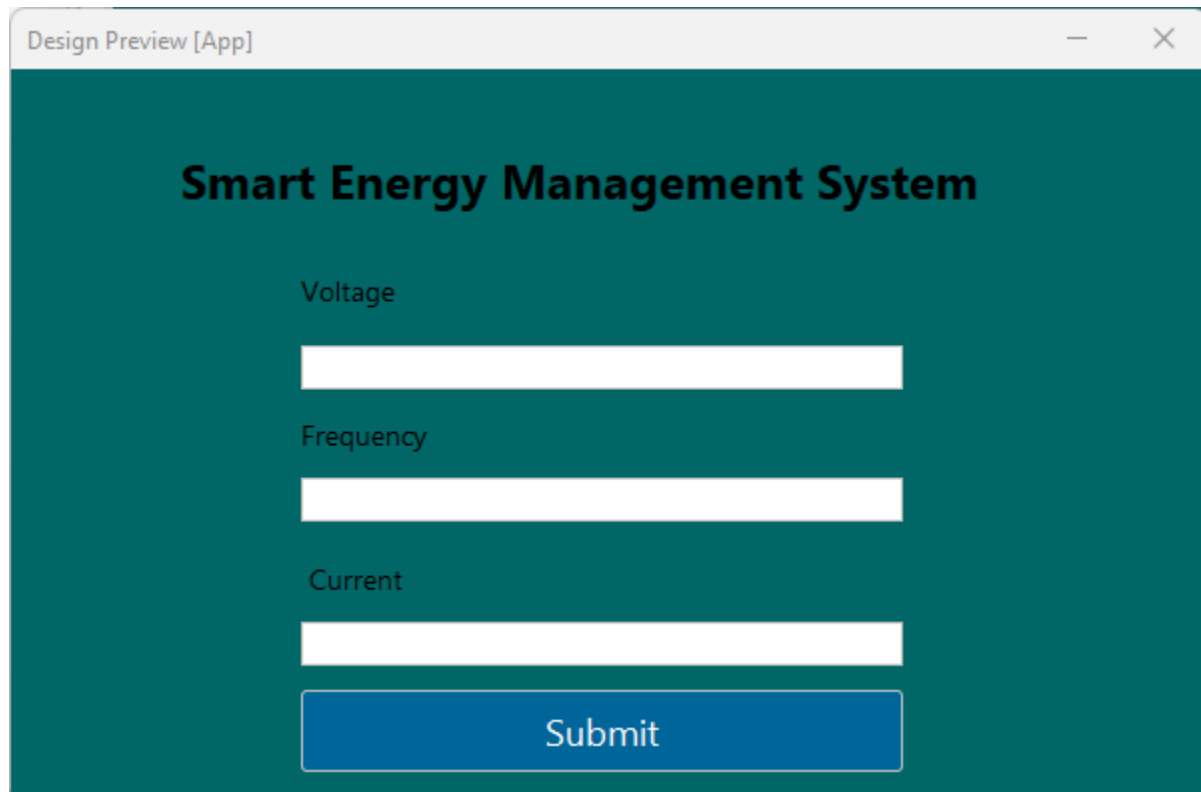
```
        this.frequency.add(frequency);
```

```
        this.current.add(current);
```

```
    }
```

```
}
```

## 7. Graphical User Interface



The image shows a web application window titled "Design Preview [App]". The main heading is "Smart Energy Management System". Below the heading, there are three input fields labeled "Voltage", "Frequency", and "Current". Each label is positioned to the left of its corresponding white input box. At the bottom of the form is a blue "Submit" button.

Design Preview [App]

### Smart Energy Management System

Voltage

Frequency

Current

Submit

## 8. Efficiency and Correctness of the solution

The solution using EJB components to handle the Smart Energy Management System (SEMS) with an IoT device has several advantages in terms of efficiency and correctness:

**Scalability:** The use of session beans and message-driven beans allows for a distributed and scalable architecture, which can handle high volumes of messaging and processing from multiple IoT devices. This makes the system efficient in terms of handling a large number of devices and data, and allows for easy scaling as the system grows.

**Reliability:** The use of entity beans and the transaction management of EJB ensures data consistency and reliability in the system, even in the case of failures or crashes. This makes the system correct and reliable in terms of data storage and retrieval, and provides a high level of fault tolerance.

**Asynchronous communication:** The use of message-driven beans and the Java Message Service (JMS) API allows for efficient and reliable communication between the IoT device and the SEMS, even in the case of intermittent network connections or delays. This ensures that data is captured and processed in a timely and efficient manner, while providing a high level of reliability.

**Timer service:** The use of the timer service allows for efficient scheduling and execution of periodic or delayed tasks in the system, such as data analytics and report generation. This ensures that the system is responsive and timely in terms of generating reports and providing real-time feedback to users.

Overall, the solution using EJB components is efficient and correct, as it provides a scalable and reliable architecture for handling the complex business logic and data processing required by the IoT device and the analytical server. By using stateless or stateful session beans, message-driven beans, entity beans, and timer service, we can ensure the efficient and reliable communication, processing, and storage of sensor data and analytical results, while providing a high level of transactional consistency and reliability.

## 9.References

[https://docs.oracle.com/cd/E24329\\_01/web.1211/e24446/ejbs.htm#INTRO315](https://docs.oracle.com/cd/E24329_01/web.1211/e24446/ejbs.htm#INTRO315)