SMART HOME ENERGY MANAGEMENT SYSTEM

Module3 – Inventory Alerts

BY Thanuja Vanjarapu
BATCH-10

RECAP: MODULE 1 & MODULE 2

Module 1 – Authentication & Access Control

Module 1 ensures that only authorized users can access the smart home system.

- Users log in using valid credentials
- The system verifies identity and user role
- Role-based access is applied (Owner, Family, Technician)
- Prevents unauthorized or duplicate logins

This module provides security and controlled access to the system.

Module 2 – Smart Device Management

Module 2 manages all smart devices connected to the home.

- Allows users to register and manage devices
- Enables turning devices ON/OFF through the system
- Maintains current device status centrally
- Sends commands through the home gateway to physical devices

This module provides centralized and controlled device operation.

Why These Modules Matter Before Module 3

- Module 1 confirms who is using the system
- Module 2 controls what devices are active
- Module 3 monitors how much energy those devices consume

NEED FOR INVENTORY ALERTS (PROBLEM STATEMENT)

In a smart home environment, multiple devices operate simultaneously and energy consumption varies dynamically throughout the day. Users are often unaware of how much energy is being consumed in real time.

Without an alert mechanism, excessive energy usage goes unnoticed until it results in power overload, system instability, or high electricity bills. Traditional systems provide consumption details only after billing cycles, leaving no opportunity for immediate corrective action.

Hence, there is a need for an inventory alert system that continuously monitors energy usage and notifies users before critical limits are crossed

CONNECTION OF MODULE 3 WITH MODULE 1 & MODULE 2

Module 3 is closely integrated with both Module 1 and Module 2 to ensure meaningful and secure energy monitoring.

Module 1 authenticates users and ensures that energy tracking is performed only for valid and authorized users. Module 2 manages smart devices and provides real-time device status, which forms the basis for energy consumption data.

Module 3 uses this device-level data from Module 2 to calculate total energy usage and generate alerts. Without authentication from Module 1 and device control from Module 2, Module 3 cannot function effectively.

Together, these modules form a complete workflow from secure access to intelligent monitoring.

PURPOSE & OBJECTIVES OF MODULE 3: INVENTORY ALERTS

Purpose of Module 3 – Inventory Alerts
The purpose of Module 3 is to continuously monitor energy consumption generated by smart devices and inform users about their usage status through timely alerts.
This module ensures that energy usage remains within safe limits by analyzing consumption data from active devices controlled in Module 2. Instead of directly controlling devices, Module 3 focuses on providing awareness and early warnings so that users can take corrective actions before critical situations occur.
Overall, Module 3 acts as a safety and awareness layer in the smart home energy management system.

Objectives of Module 3
The objectives of Module 3 are:
- To track real-time energy consumption of active devices
- To calculate total and remaining energy usage for users
- To classify energy usage into normal, warning, and critical levels
- To generate alerts when usage crosses predefined thresholds
- To inform users early so they can reduce or manage energy consumption
- To prevent sudden overloads and system instability

These objectives ensure efficient energy management and safe system operation.

ARCHITECTURE & PROBLEMS SOLVED BY MODULE 3

Architecture of Module 3 – Real-Time Energy Tracking

Module 3 follows a layered architecture to ensure accurate energy monitoring and alert generation.

- The **Controller layer** receives requests related to energy usage from the user interface.
- The **Service layer** processes device-wise energy data received from Module 2 and evaluates usage levels.
- The **Repository layer** stores and retrieves energy usage records.
- The **Model layer** represents energy usage data generated during device operations.

This architecture allows Module 3 to operate independently while relying on authenticated users (Module 1) and active devices (Module 2).

Problems Solved by Module 3

Module 3 addresses critical issues related to uncontrolled energy consumption in smart homes:

- Users are unaware of real-time energy usage
- No early warning before energy limits are crossed
- Sudden power overloads due to multiple active devices
- Difficulty in identifying risky usage patterns
- Dependence on post-billing information instead of live alerts

By continuously tracking energy usage and generating alerts, Module 3 helps users respond before critical situations occur.

ACTIVITIES OF MODULE3

1. Device-wise Energy Data Collection
Module 3 collects energy consumption data from all active devices that are currently being controlled through Module 2. Only devices that are ON contribute to energy tracking.

2. Continuous Usage Tracking
The system continuously tracks how much energy each device consumes over time, ensuring that usage is monitored in real time rather than after billing.

3. **Usage Record Creation**
For every tracking cycle, Module 3 creates usage records that store energy consumption details along with time information. These records help in understanding usage patterns.

4. **Total Energy Calculation**
The system calculates the total energy consumed by aggregating device-wise usage data for the authenticated user.

5. Threshold Comparison
The calculated energy usage is compared with predefined energy limits to determine whether consumption is within safe boundaries.

6. Usage Status Identification
Based on the comparison, the system categorizes energy usage into:

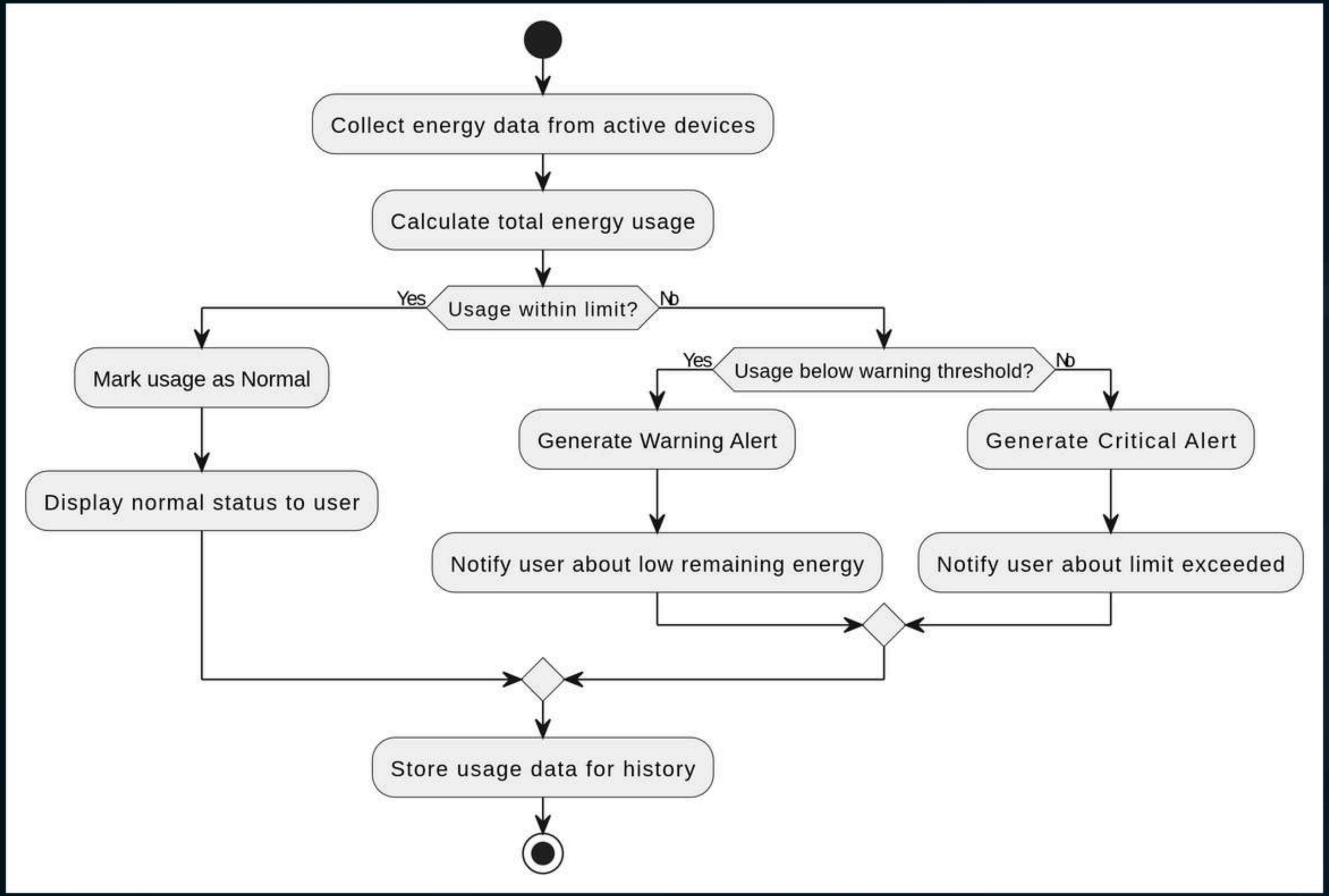
- Normal usage
- Warning level usage
- Critical usage

7. **Alert Triggering**
When usage crosses warning or critical thresholds, Module 3 automatically triggers alerts to inform the user.

8. User Notification Display
The generated alerts are sent to the user interface so that users are immediately aware of their current energy status.

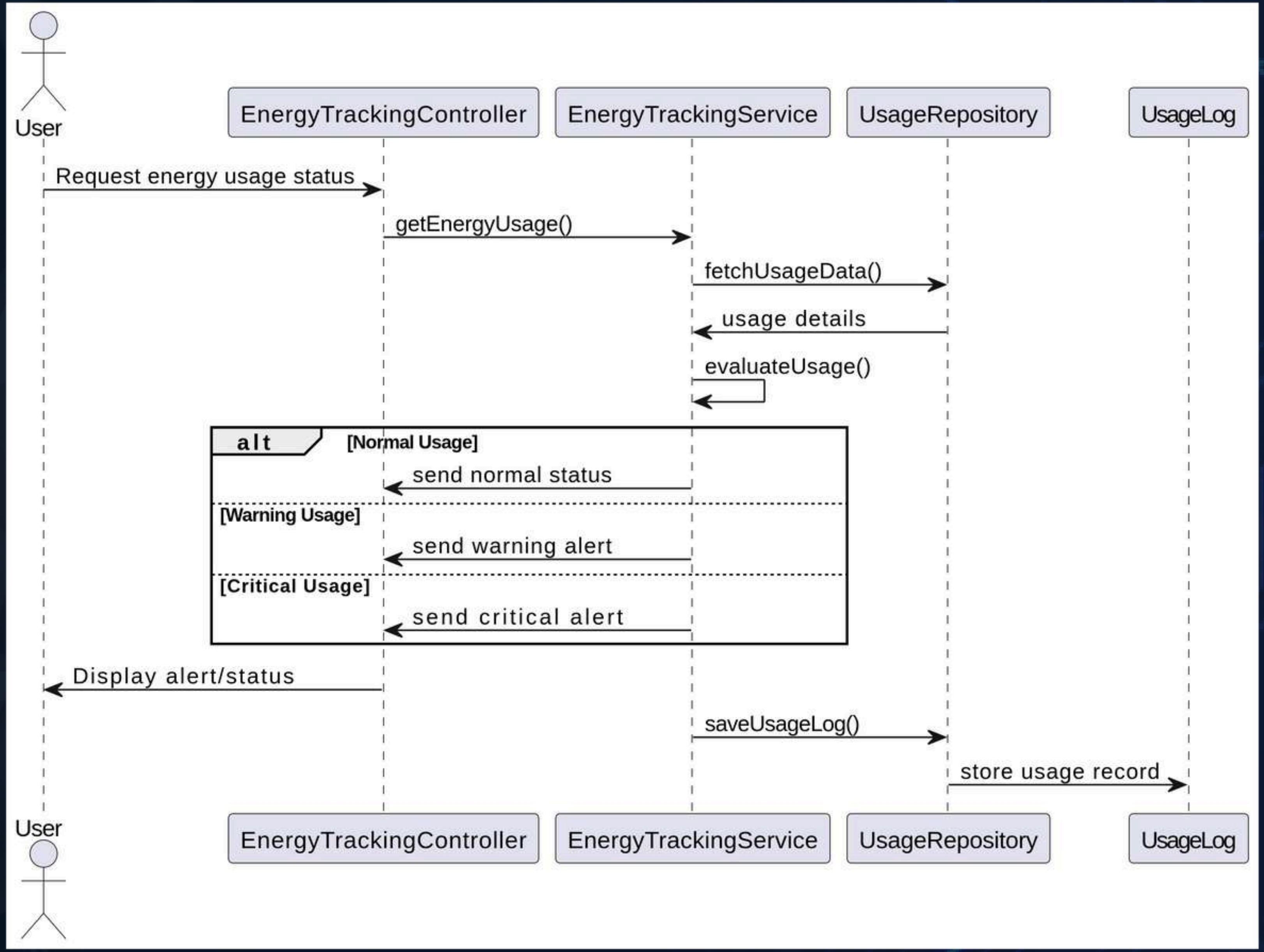
9. Usage History Storage
Module 3 stores historical energy usage data, which can later be used for analytics, reports, and decision making.

ACTIVITY AND SEQUENCE DIAGRAMS OF MODULE 3



Collect energy data from active devices
Calculate total energy usage
Usage within limit?
Yes
No
Mark usage as Normal
Display normal status to user
Usage below warning threshold?
Yes
No
Generate Warning Alert
Generate Critical Alert
Notify user about low remaining energy
Notify user about limit exceeded
Store usage data for history

Figure(i)



Figure(ii)

CLASSES OF MODULE3

Module 3 is responsible for tracking and storing energy consumption data generated by smart devices in real time.

1. UsageLog (Model Class)
Role: Represents energy usage data of a device.
- Stores device-wise energy consumption
- Maintains timestamp of usage
- Acts as the core data entity for energy tracking

This class creates the usage table in the database.

2. EnergyTrackingController (Controller Class)
Role: Entry point for energy tracking requests.
- Receives requests for live and historical energy data
- Handles API calls from UI or dashboard
- Forwards requests to the service layer

This class does not contain business logic.

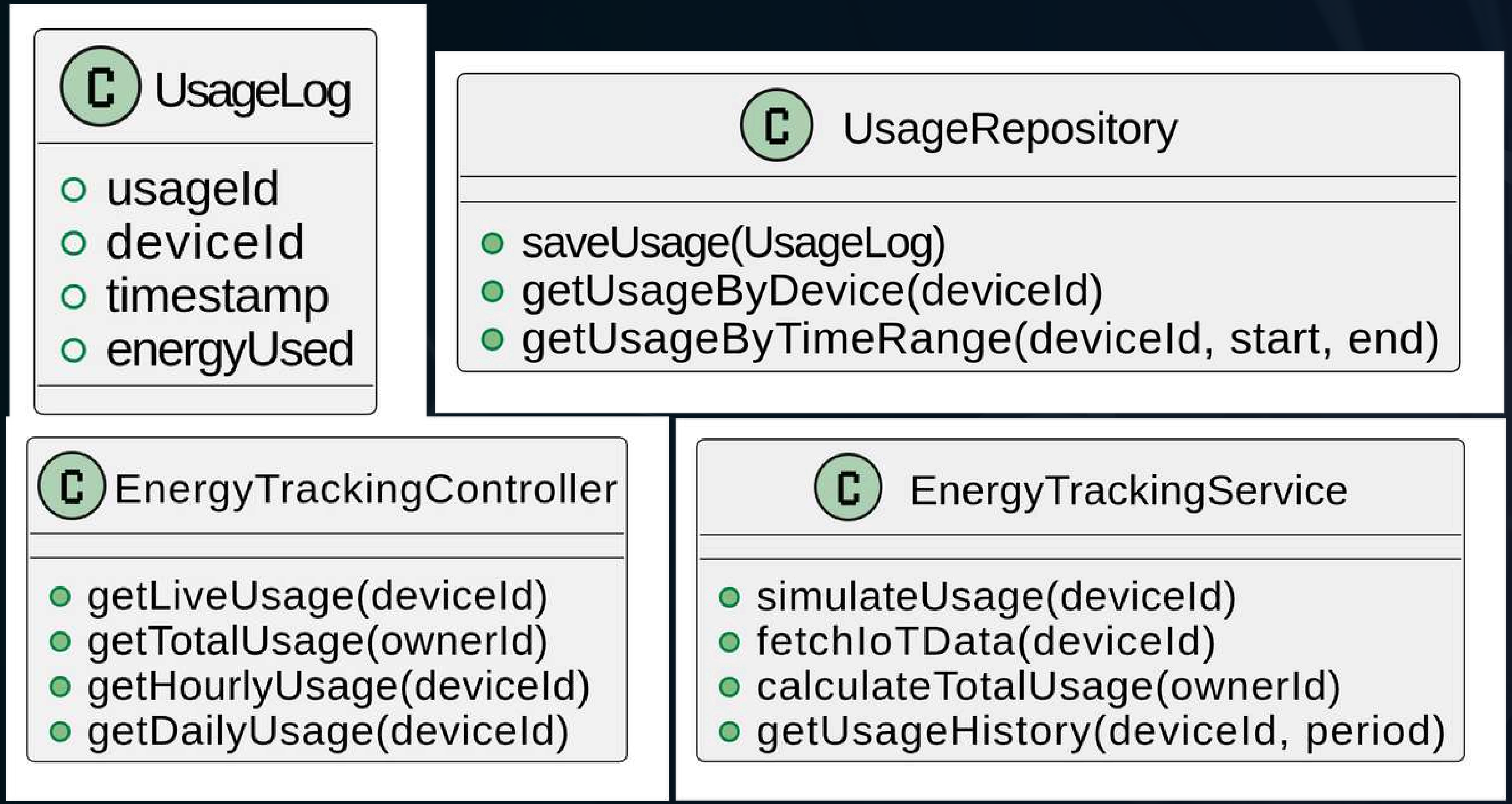
3. EnergyTrackingService (Service Class)
Role: Core business logic of Module 3.
- Calculates energy consumption
- Simulates or fetches real-time usage
- Aggregates usage data for reports

This class decides how energy is calculated and processed.

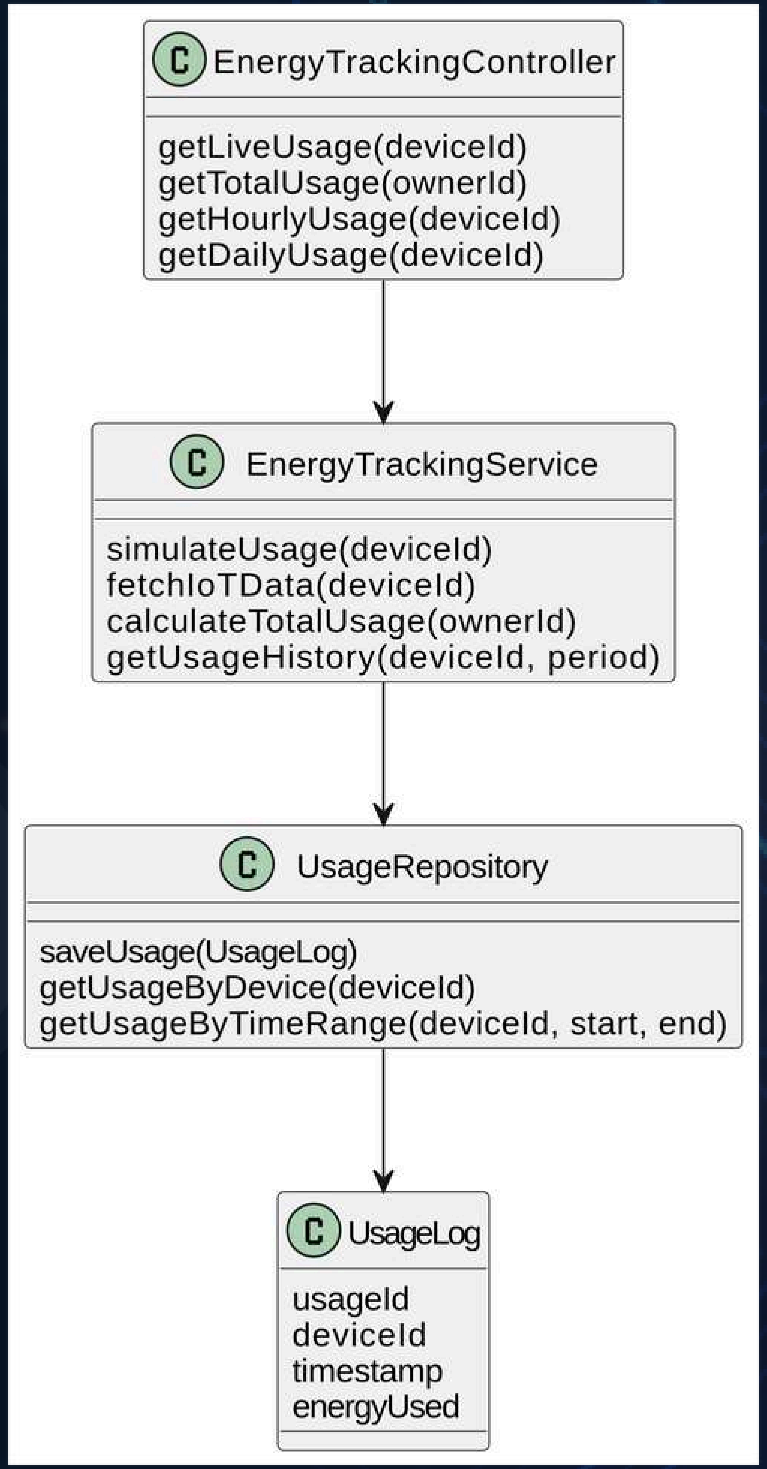
4. UsageRepository (Repository Class)
Role: Database interaction for usage data.
- Saves energy usage logs
- Retrieves usage history
- Fetches data for analytics and alerts

This class handles only database operations.

CLASS DIAGRAMS FOR MODULE 3

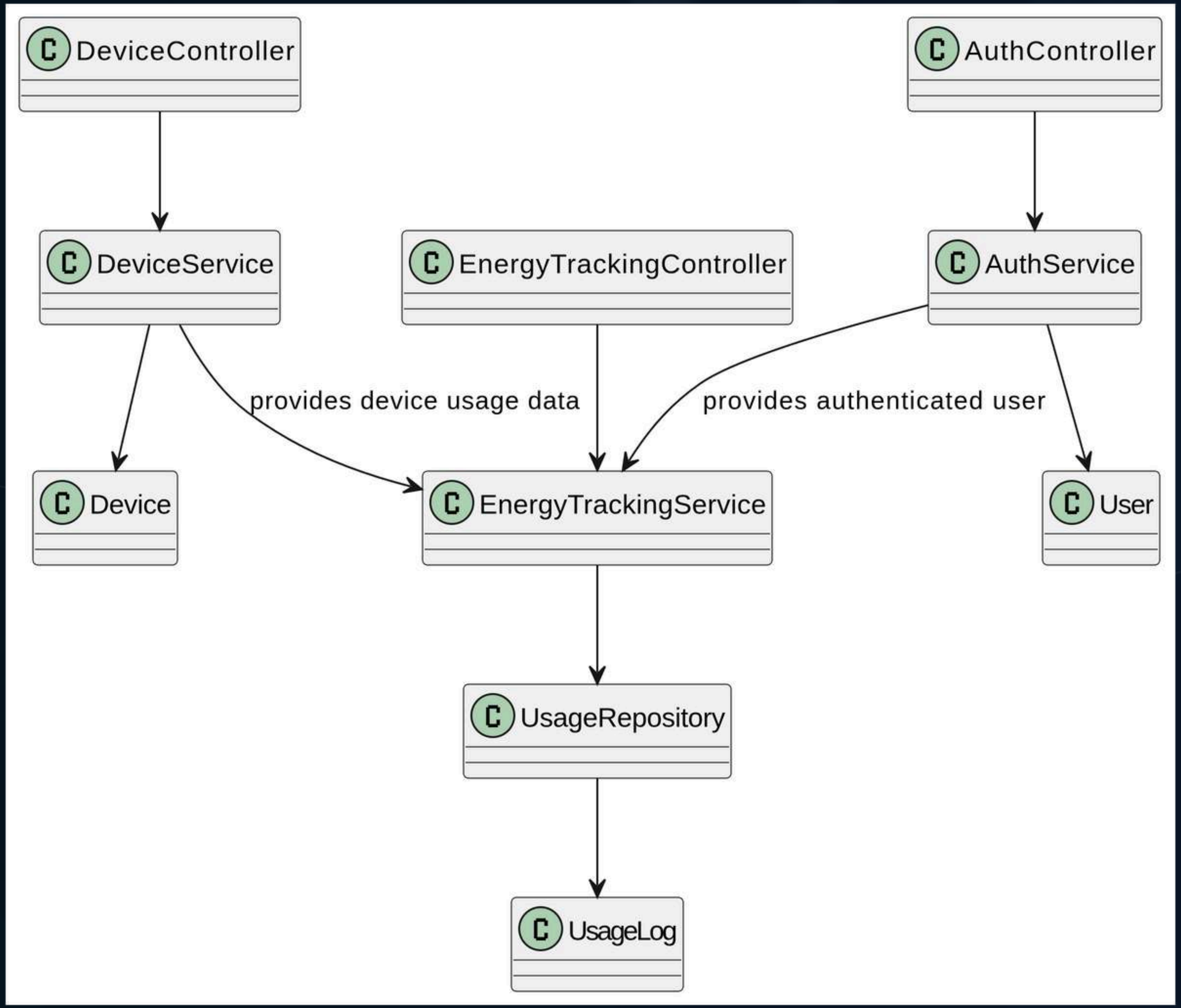


Figure(i)



Figure(ii)

CLASS DIAGRAM FOR MODULE 3 AND ITS CONNECTION WITH MODULE 1,2



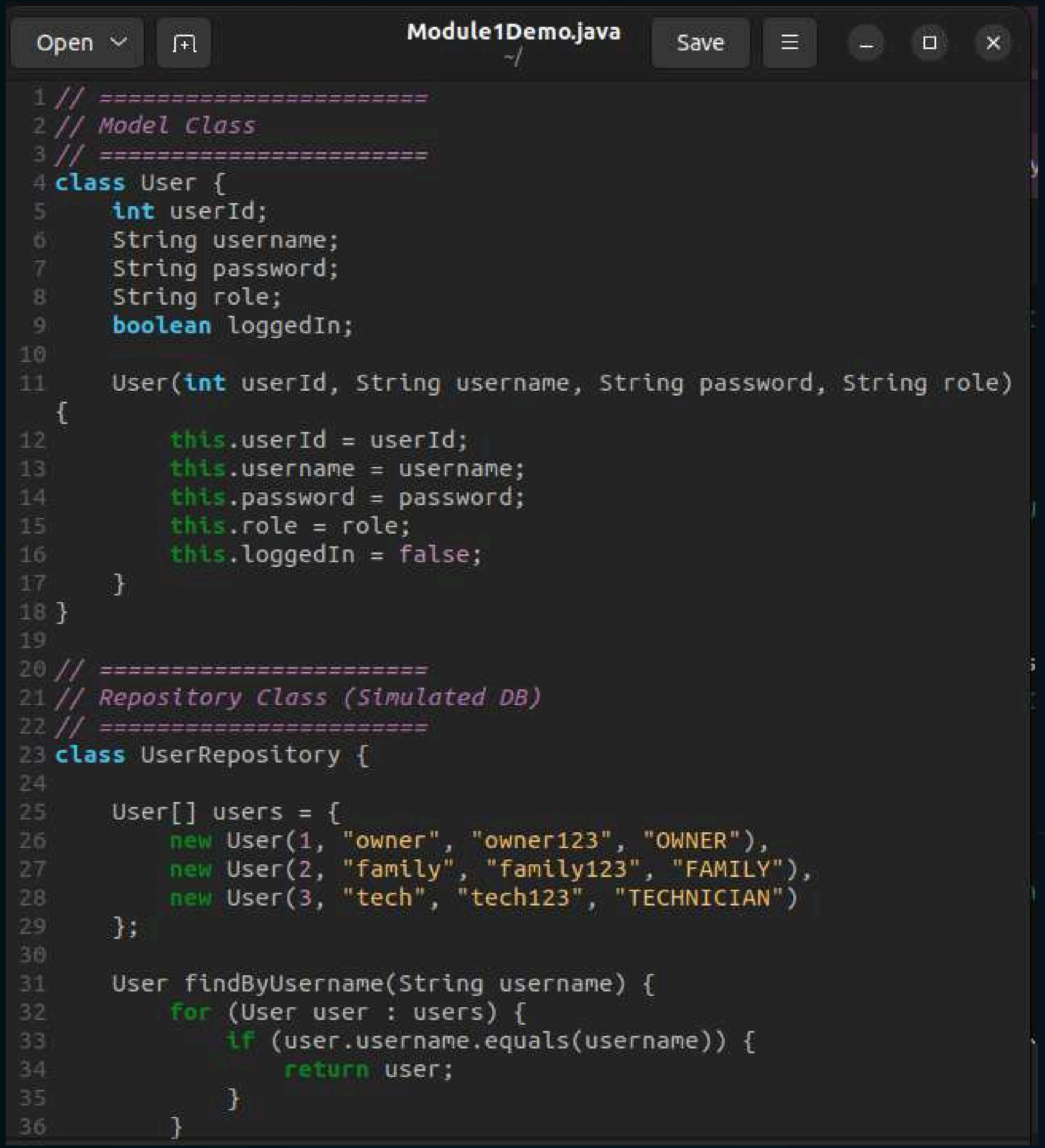
Figure(iii)

DB SCHEMA FOR MODULE 3

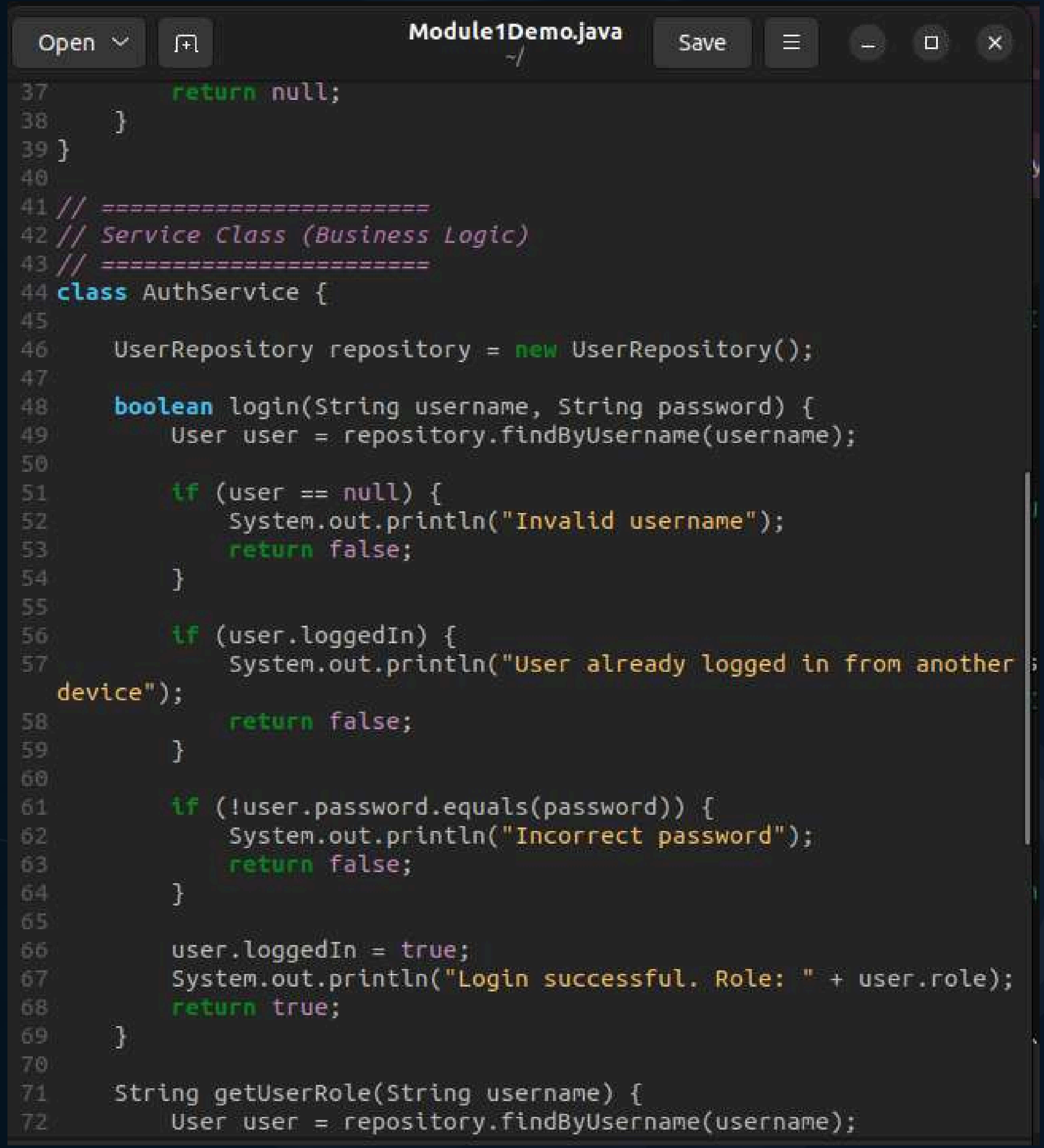
```
CREATE TABLE usage_logs (
    usage_id BIGINT PRIMARY KEY AUTO_INCREMENT,
    device_id BIGINT NOT NULL,            -- Device whose energy is tracked
    energy_used FLOAT NOT NULL,           -- Energy consumed (in units / kWh)
    recorded_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT fk_device_usage
        FOREIGN KEY (device_id)
        REFERENCES devices(device_id)
        ON DELETE CASCADE
);
```

- usage_logs table maps directly to UsageLog model class
- device_id links Module 3 with Module 2 (devices table)
- Time-based tracking is supported using recorded_at
- ON DELETE CASCADE ensures:If a device is removed, its usage logs are also removed

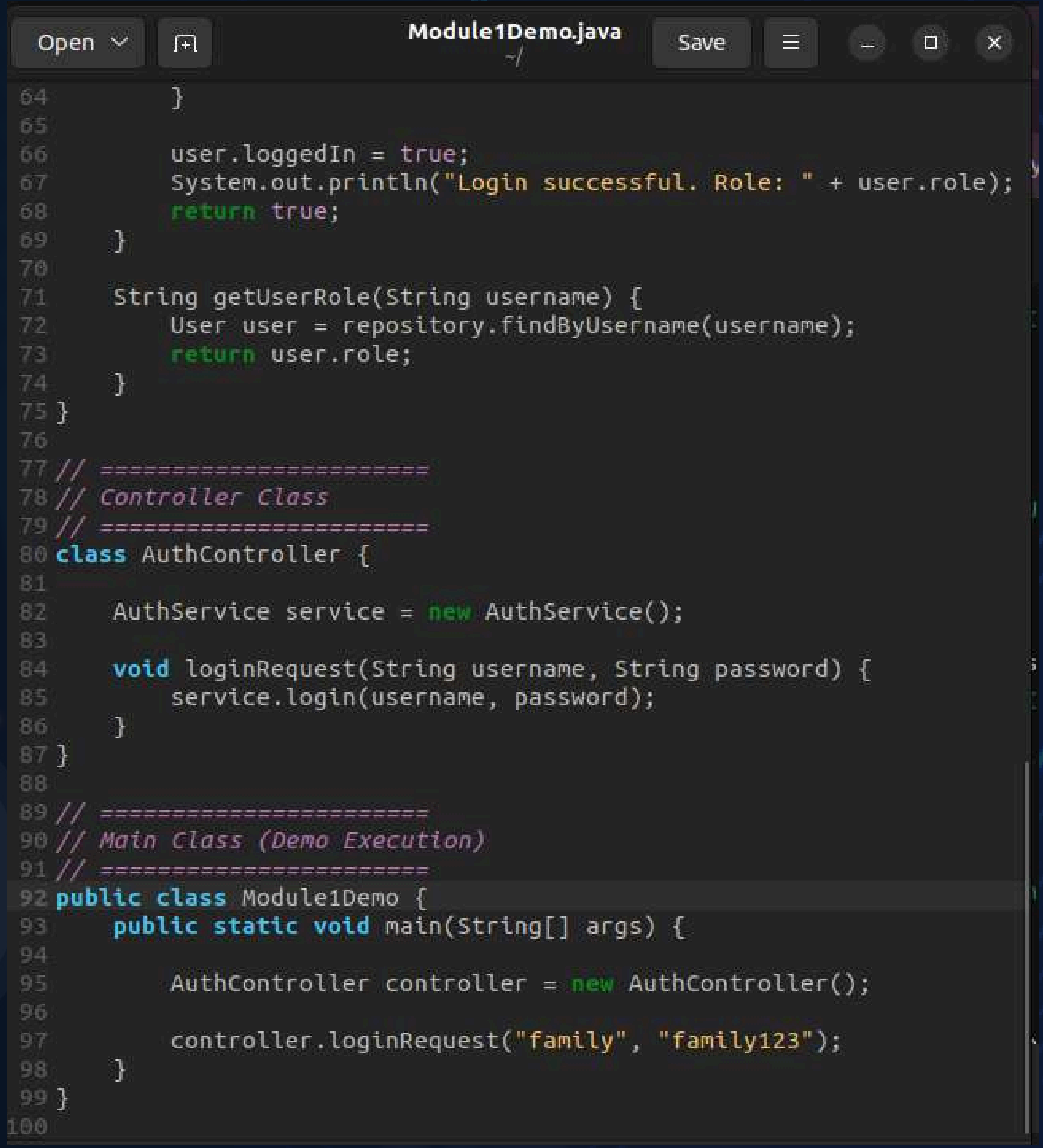
MODULE1 DEMO SCREENSHOTS



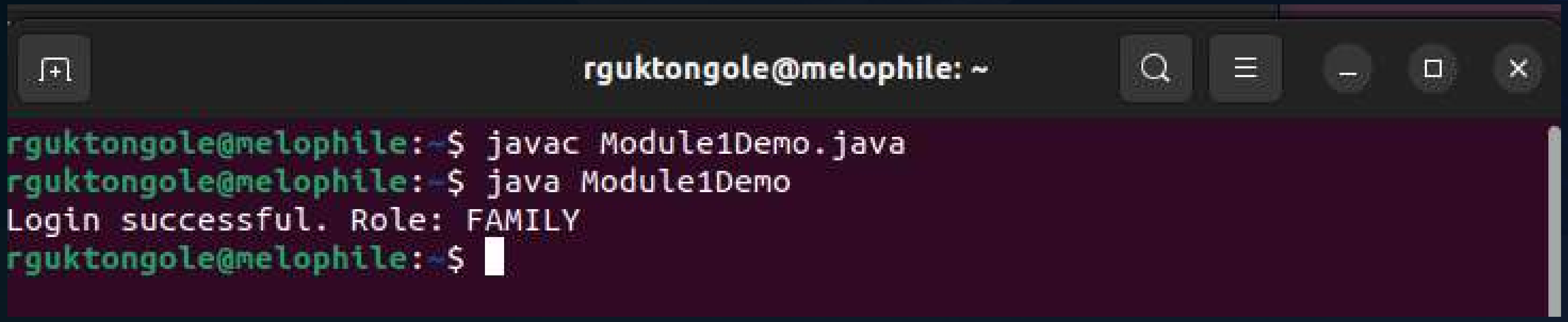
```java
// ======================
// Model Class
// ======================
class User {
    int userId;
    String username;
    String password;
    String role;
    boolean loggedIn;

    User(int userId, String username, String password, String role)
 {
        this.userId = userId;
        this.username = username;
        this.password = password;
        this.role = role;
        this.loggedIn = false;
    }
}

// ======================
// Repository Class (Simulated DB)
// ======================
class UserRepository {

    User[] users = {
        new User(1, "owner", "owner123", "OWNER"),
        new User(2, "family", "family123", "FAMILY"),
        new User(3, "tech", "tech123", "TECHNICIAN")
    };

    User findByUsername(String username) {
        for (User user : users) {
            if (user.username.equals(username)) {
                return user;
            }
        }
```



```java
            return null;
    }
}

// ======================
// Service Class (Business Logic)
// ======================
class AuthService {

    UserRepository repository = new UserRepository();

    boolean login(String username, String password) {
        User user = repository.findByUsername(username);

        if (user == null) {
            System.out.println("Invalid username");
            return false;
        }

        if (user.loggedIn) {
            System.out.println("User already logged in from another
device");
            return false;
        }

        if (!user.password.equals(password)) {
            System.out.println("Incorrect password");
            return false;
        }

        user.loggedIn = true;
        System.out.println("Login successful. Role: " + user.role);
        return true;
    }

    String getUserRole(String username) {
        User user = repository.findByUsername(username);
```

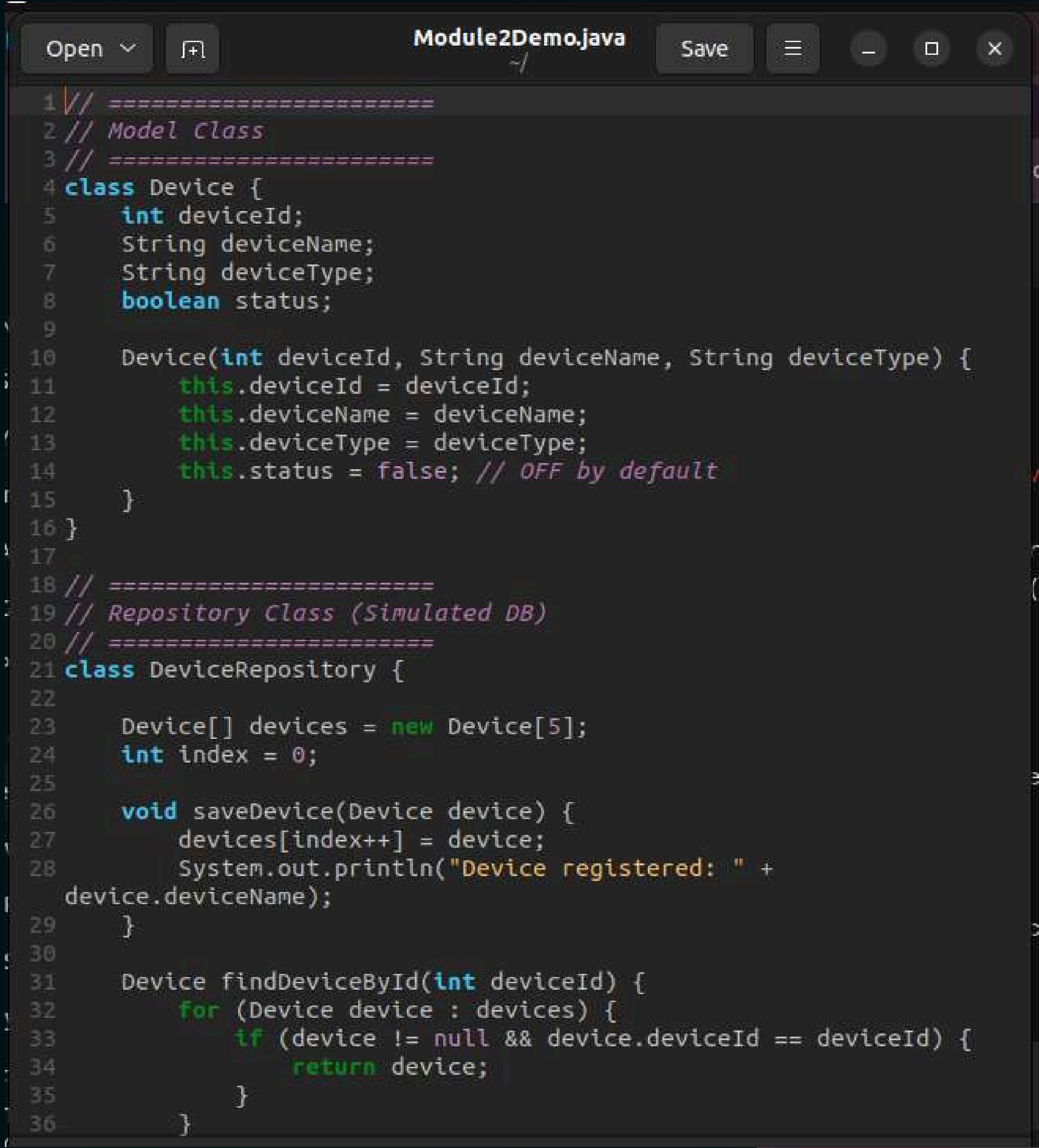


```java
        }

        user.loggedIn = true;
        System.out.println("Login successful. Role: " + user.role);
        return true;
    }

    String getUserRole(String username) {
        User user = repository.findByUsername(username);
        return user.role;
    }
}

// ======================
// Controller Class
// ======================
class AuthController {

    AuthService service = new AuthService();

    void loginRequest(String username, String password) {
        service.login(username, password);
    }
}

// ======================
// Main Class (Demo Execution)
// ======================
public class Module1Demo {
    public static void main(String[] args) {

        AuthController controller = new AuthController();

        controller.loginRequest("family", "family123");
    }
}

```

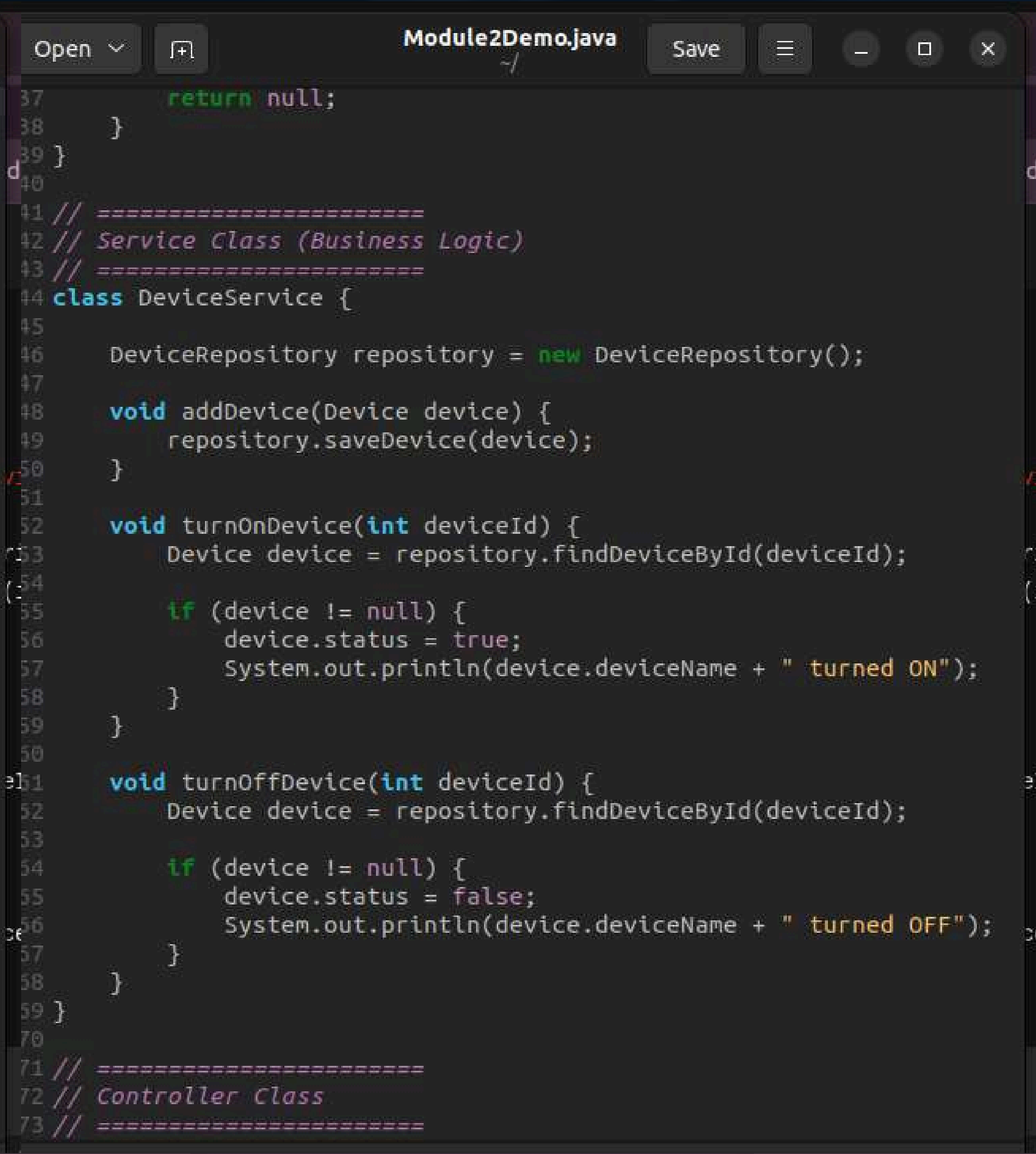


```
rguktongole@melophile:~$ javac Module1Demo.java
rguktongole@melophile:~$ java Module1Demo
Login successful. Role: FAMILY
rguktongole@melophile:~$ 
```

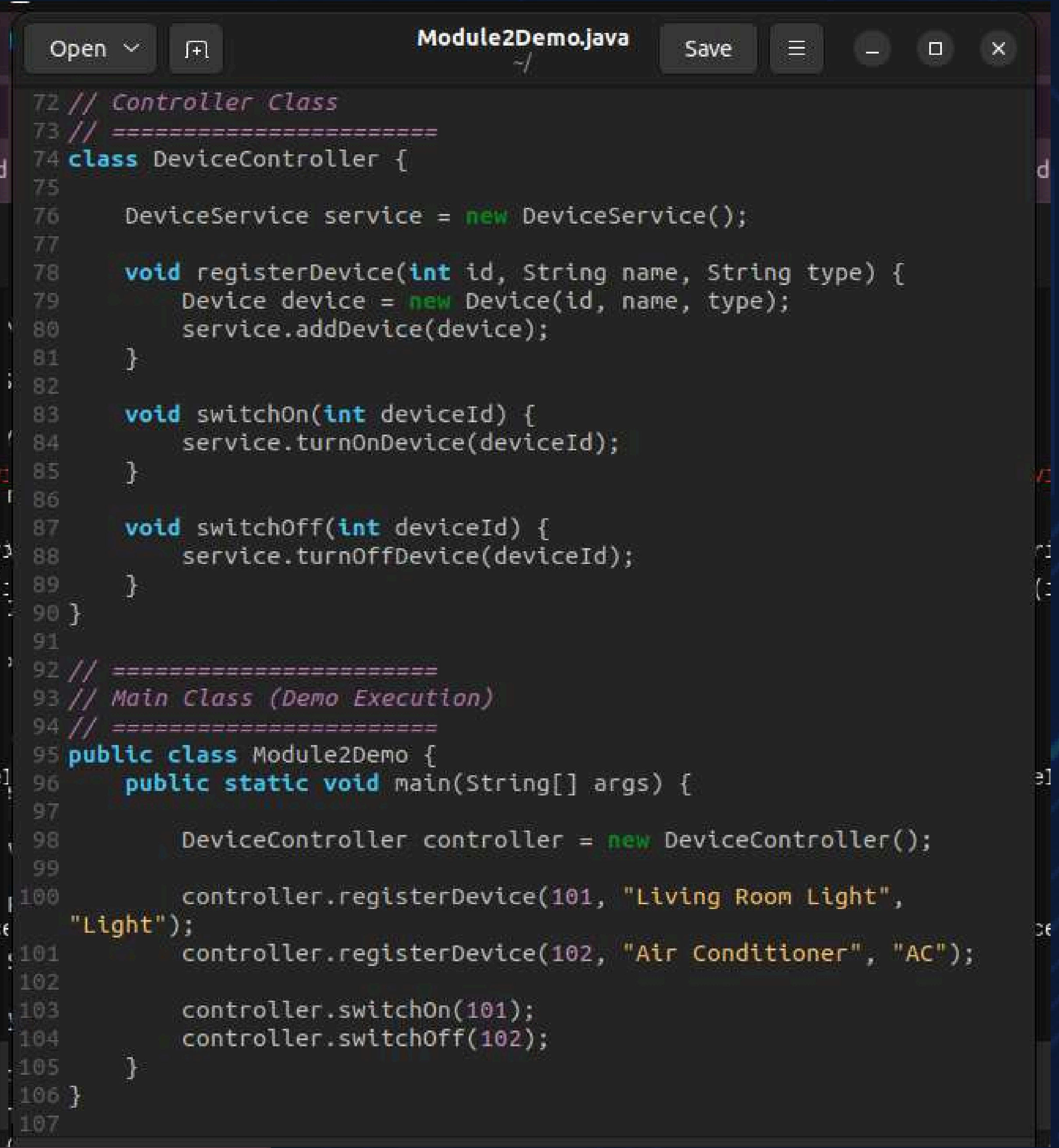
MODULE2 DEMO SCREENSHOTS



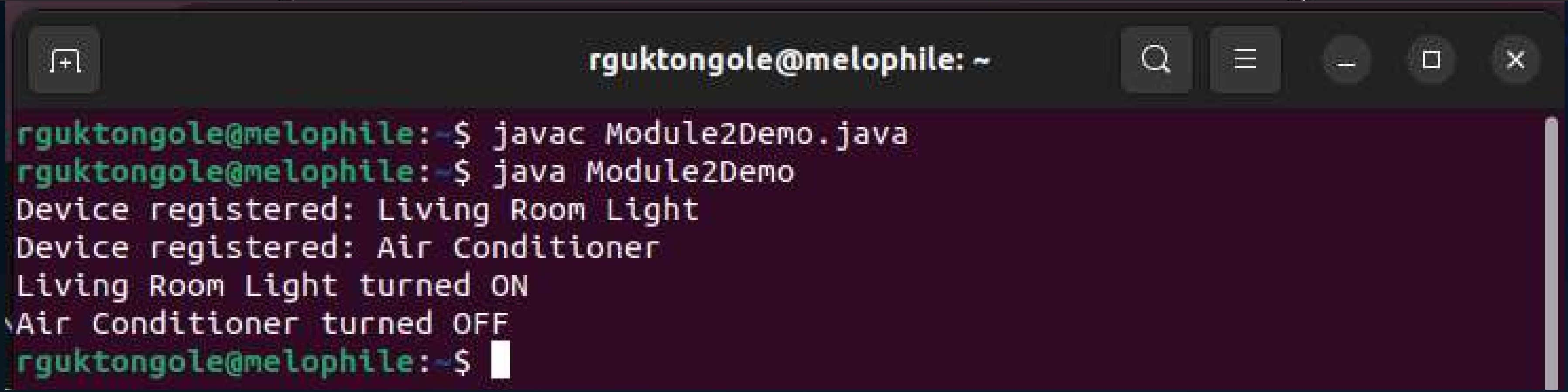
```java
// ======================
// Model Class
// ======================
class Device {
    int deviceId;
    String deviceName;
    String deviceType;
    boolean status;

    Device(int deviceId, String deviceName, String deviceType) {
        this.deviceId = deviceId;
        this.deviceName = deviceName;
        this.deviceType = deviceType;
        this.status = false; // OFF by default
    }
}

// ======================
// Repository Class (Simulated DB)
// ======================
class DeviceRepository {

    Device[] devices = new Device[5];
    int index = 0;

    void saveDevice(Device device) {
        devices[index++] = device;
        System.out.println("Device registered: " +
device.deviceName);
    }

    Device findDeviceById(int deviceId) {
        for (Device device : devices) {
            if (device != null && device.deviceId == deviceId) {
                return device;
            }
        }
```



```java
        return null;
    }
}

// ======================
// Service Class (Business Logic)
// ======================
class DeviceService {

    DeviceRepository repository = new DeviceRepository();

    void addDevice(Device device) {
        repository.saveDevice(device);
    }

    void turnOnDevice(int deviceId) {
        Device device = repository.findDeviceById(deviceId);

        if (device != null) {
            device.status = true;
            System.out.println(device.deviceName + " turned ON");
        }
    }

    void turnOffDevice(int deviceId) {
        Device device = repository.findDeviceById(deviceId);

        if (device != null) {
            device.status = false;
            System.out.println(device.deviceName + " turned OFF");
        }
    }
}

// ======================
// Controller Class
// ======================
```

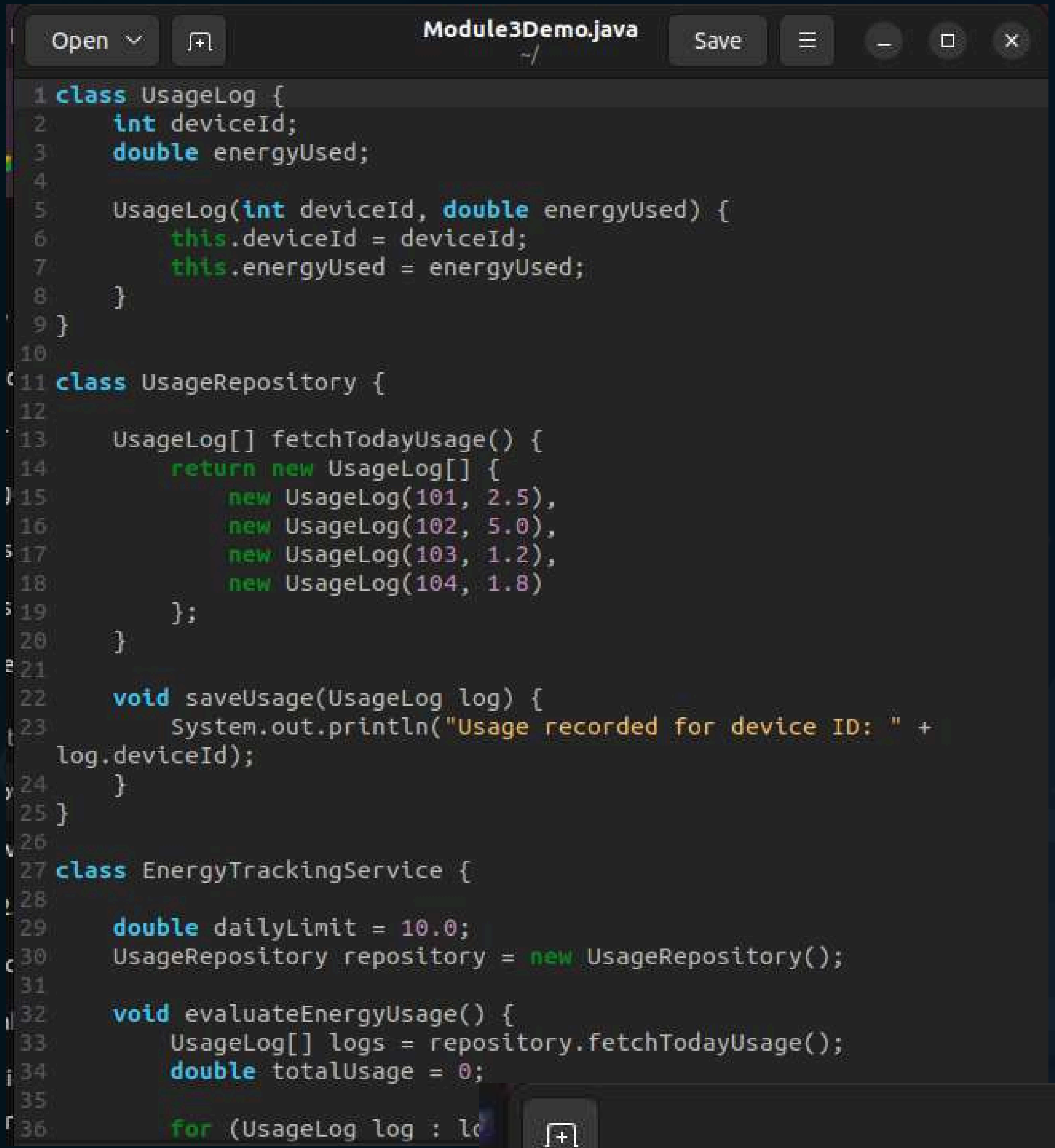


```java
// Controller Class
// ======================
class DeviceController {

    DeviceService service = new DeviceService();

    void registerDevice(int id, String name, String type) {
        Device device = new Device(id, name, type);
        service.addDevice(device);
    }

    void switchOn(int deviceId) {
        service.turnOnDevice(deviceId);
    }

    void switchOff(int deviceId) {
        service.turnOffDevice(deviceId);
    }
}

// ======================
// Main Class (Demo Execution)
// ======================
public class Module2Demo {
    public static void main(String[] args) {

        DeviceController controller = new DeviceController();

        controller.registerDevice(101, "Living Room Light",
"Light");
        controller.registerDevice(102, "Air Conditioner", "AC");

        controller.switchOn(101);
        controller.switchOff(102);
    }
}
```

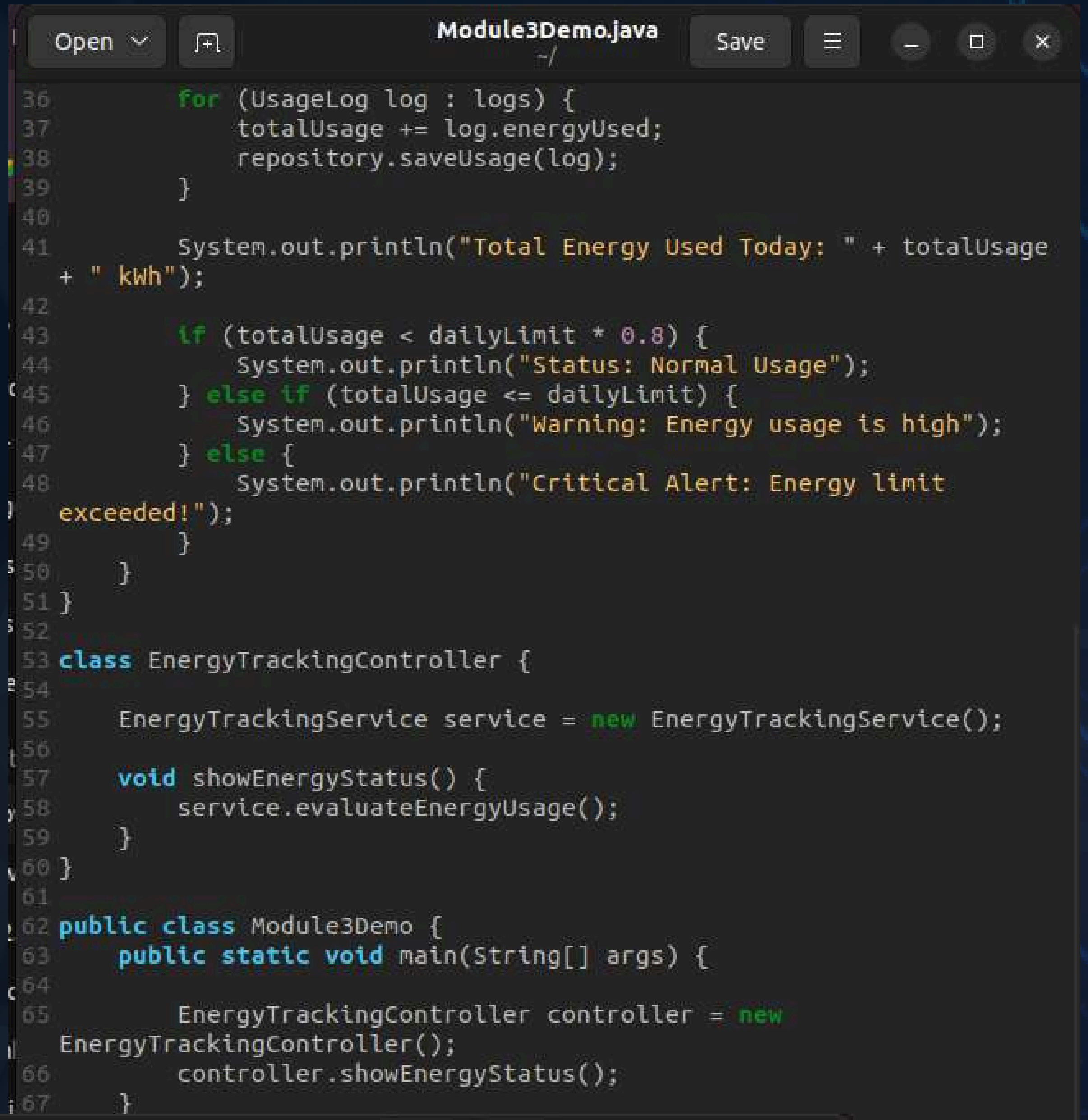


```
rguktongole@melophile:~$ javac Module2Demo.java
rguktongole@melophile:~$ java Module2Demo
Device registered: Living Room Light
Device registered: Air Conditioner
Living Room Light turned ON
Air Conditioner turned OFF
rguktongole@melophile:~$
```

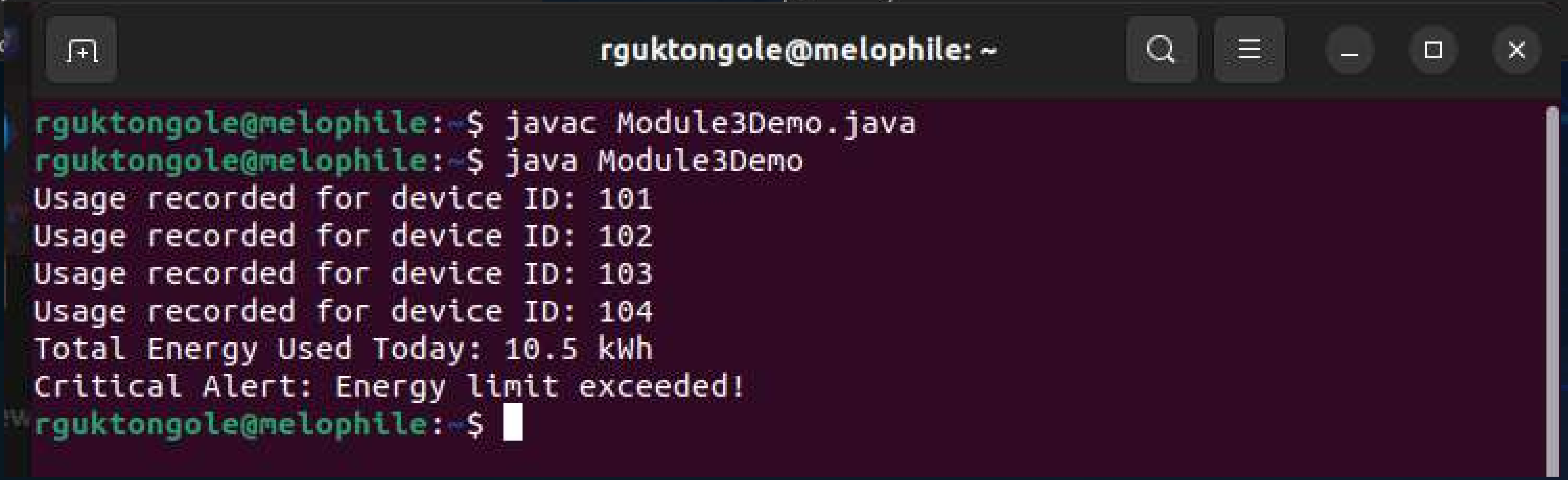
MODULE3 DEMO SCREENSHOTS



```java
class UsageLog {
    int deviceId;
    double energyUsed;

    UsageLog(int deviceId, double energyUsed) {
        this.deviceId = deviceId;
        this.energyUsed = energyUsed;
    }
}

class UsageRepository {

    UsageLog[] fetchTodayUsage() {
        return new UsageLog[] {
            new UsageLog(101, 2.5),
            new UsageLog(102, 5.0),
            new UsageLog(103, 1.2),
            new UsageLog(104, 1.8)
        };
    }

    void saveUsage(UsageLog log) {
        System.out.println("Usage recorded for device ID: " +
log.deviceId);
    }
}

class EnergyTrackingService {

    double dailyLimit = 10.0;
    UsageRepository repository = new UsageRepository();

    void evaluateEnergyUsage() {
        UsageLog[] logs = repository.fetchTodayUsage();
        double totalUsage = 0;

        for (UsageLog log : lo
```



```java
        for (UsageLog log : logs) {
            totalUsage += log.energyUsed;
            repository.saveUsage(log);
        }

        System.out.println("Total Energy Used Today: " + totalUsage
+ " kWh");

        if (totalUsage < dailyLimit * 0.8) {
            System.out.println("Status: Normal Usage");
        } else if (totalUsage <= dailyLimit) {
            System.out.println("Warning: Energy usage is high");
        } else {
            System.out.println("Critical Alert: Energy limit
exceeded!");
        }
    }
}

class EnergyTrackingController {

    EnergyTrackingService service = new EnergyTrackingService();

    void showEnergyStatus() {
        service.evaluateEnergyUsage();
    }
}

public class Module3Demo {
    public static void main(String[] args) {

        EnergyTrackingController controller = new
EnergyTrackingController();
        controller.showEnergyStatus();
    }
```



```
rguktongole@melophile:~$ javac Module3Demo.java
rguktongole@melophile:~$ java Module3Demo
Usage recorded for device ID: 101
Usage recorded for device ID: 102
Usage recorded for device ID: 103
Usage recorded for device ID: 104
Total Energy Used Today: 10.5 kWh
Critical Alert: Energy limit exceeded!
rguktongole@melophile:~$
```

THANK YOU !