Contents

Project File Structure

```
iot-smart-home-dashboard/
├── src/
│   ├── main/
│   │   ├── java/com/smarthome/
│   │   │   ├── model/                    # Entity Models (4 files)
│   │   │   │   ├── Customer.java         # User entity with security (343 lines)
│   │   │   │   ├── DeletedDeviceEnergyRecord.java  # Energy preservation
│   │   │   │   ├── DevicePermission.java    # Group access control
│   │   │   │   └── Gadget.java           # Device entity with energy tracking (275 lines)
│   │   │   │
│   │   │   ├── service/                  # Business Logic Services (8 files)
│   │   │   │   ├── CalendarEventService.java     # Event scheduling
│   │   │   │   ├── CustomerService.java          # User management with security (464 lines)
│   │   │   │   ├── DeviceHealthService.java      # Device health monitoring
│   │   │   │   ├── EnergyManagementService.java  # Energy calculations (255 lines)
│   │   │   │   ├── GadgetService.java            # Device management
│   │   │   │   ├── SmartHomeService.java         # Main orchestrator (1,606 lines)
│   │   │   │   ├── SmartScenesService.java       # Scene automation
│   │   │   │   ├── TimerService.java             # Background automation (400 lines)
│   │   │   │   └── WeatherService.java           # Weather integration
│   │   │   │
│   │   │   ├── util/                     # Utility Classes (2 files)
│   │   │   │   ├── DynamoDBConfig.java      # Database configuration
│   │   │   │   └── SessionManager.java      # Session state management
│   │   │   │
│   │   │   └── SmartHomeDashboard.java      # Main application entry point (3,500+ lines)
│   │   │
│   │   └── resources/
│   │       └── application.properties       # Application configuration
│   │
│   └── test/java/com/smarthome/          # Unit Tests (2 files)
│       ├── DevicePermissionTest.java        # Device permission testing
│       └── SmartHomeServiceTest.java        # Service layer testing
│
├── target/                               # Maven Build Directory
│   ├── classes/                          # Compiled Java classes
│   ├── generated-sources/                # Auto-generated sources
│   └── maven-status/                     # Build status
│
├── Build & Configuration Files (4 files)
│   ├── pom.xml                           # Maven project configuration
│   ├── dependency-reduced-pom.xml        # Shaded JAR dependencies
│   └── application.properties            # Runtime configuration
```

Key Architecture Components

Model Layer (`src/main/java/com/smarthome/model/`)
- Customer.java: User entity with enhanced security features, group management, and device permissions
- Gadget.java: Smart device entity with real-time energy tracking and timer automation
- DeletedDeviceEnergyRecord.java: Preserves energy consumption data for accurate billing
- DevicePermission.java: Manages group-based device access control

Service Layer (`src/main/java/com/smarthome/service/`)
- SmartHomeService.java: Main orchestrator coordinating all business operations
- CustomerService.java: User management with progressive security lockout system
- EnergyManagementService.java: Indian electricity tariff-based cost calculations
- TimerService.java: Background automation with concurrent timer execution
- Additional Services: Calendar, Weather, Health, Scenes, Gadget management

Utility Layer (`src/main/java/com/smarthome/util/`)
- DynamoDBConfig.java: Database connection and configuration management
- SessionManager.java: Thread-safe user session state management

Main Application
- SmartHomeDashboard.java: Entry point with 19 menu options and comprehensive UI

Database & Infrastructure

- Database: Amazon DynamoDB (Local & AWS)
- Security: BCrypt password hashing, progressive lockout policies
- Concurrency: Thread-safe singleton services with background automation
- Energy Management: Real-time consumption tracking with Indian tariff calculations

---

How to Run the Application

Quick Start (Recommended)
Option 1: Automated Startup (Windows)
Navigate to project directory

```
cd C:\Users\gochendr\Downloads\atlas
```

1. Start DynamoDB Local Server (Terminal 1)
start-dynamodb.bat

2. Start IoT Smart Home Dashboard (Terminal 2)
QUICK_START.bat

Option 2: Manual Startup (Cross-platform)
Step 1: Start DynamoDB Local Server
Navigate to DynamoDB directory

```
cd C:\Users\gochendr\Downloads\atlas\dynamodb-local
```

Start DynamoDB Local on port 8002

```
java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -sharedDb -port
8002
```

Step 2: Build and Run Application

Navigate to application directory

```
cd C:\Users\gochendr\Downloads\atlas\iot-smart-home-dashboard
```

Option A: Run with Maven (Development)

```
mvn compile
mvn exec:java -Dexec.mainClass="com.smarthome.SmartHomeDashboard"
```

Option B: Build JAR and Run (Production)

```
mvn clean package -Dmaven.test.skip=true
java -jar target\iot-smart-home-dashboard-1.0.0.jar
```

Prerequisites

Required Software

- Java 21 LTS or later
- Apache Maven 3.9.11 or later
- DynamoDB Local (included in project)

Detailed Startup Process

1. DynamoDB Local Server Setup

Command breakdown:

```
java -Djava.library.path=./DynamoDBLocal_lib     Native library path
     -jar DynamoDBLocal.jar                      DynamoDB Local JAR
     -sharedDb                                   Shared database mode
     -port 8002                                  Custom port (default: 8000)
```

Expected output:

Initializing DynamoDB Local with the following configuration:

```
Port: 8002
InMemory: false
DbPath: null
SharedDb: true
shouldDelayTransientStatuses: false
CorsParams:
```

2. Application Build Process

Clean build (if needed)

```
mvn clean
```

Compile source code

```
mvn compile
```

Run tests (optional)

```
mvn test
```

Package application with dependencies

```
mvn package -Dmaven.test.skip=true
```

This creates:
- target/iot-smart-home-dashboard-1.0.0.jar (shaded JAR)
- target/original-iot-smart-home-dashboard-1.0.0.jar (original)

Application Execution Options

Development Mode (Maven Exec Plugin)

```
mvn exec:java -Dexec.mainClass="com.smarthome.SmartHomeDashboard"
```

Advantages: Fast restart, debug-friendly
Use for: Development and testing

Production Mode (Standalone JAR)

```
java -jar target/iot-smart-home-dashboard-1.0.0.jar
```

Advantages: No Maven dependency, faster startup
Use for: Production deployment

Troubleshooting

Common Issues and Solutions

Port 8002 Already in Use

Check what's using the port

```
netstat -an | findstr :8002
```

Kill Java processes (Windows)

```
taskkill /F /IM java.exe
```

Build Failures

Force clean and rebuild

```
mvn clean package -Dmaven.test.skip=true -U
```

Clear Maven repository cache

```
mvn dependency:purge-local-repository
```

Database Connection Issues

Verify DynamoDB Local is running

```
curl http://localhost:8002/
```

Check application.properties

```
src/main/resources/application.properties
```

Java Version Issues

Check Java version

```
java -version
```

Should show: openjdk version "21.x.x" or later
Update JAVA_HOME if needed

Application Features Available After Startup

Once both services are running, you'll have access to:

🔐 Authentication System
- User registration with enhanced security validation
- Progressive lockout system (2→5min, 3→15min, 5→1hr, 7→4hr, 10→24hr)
- BCrypt password hashing with common password blocking

🏠 Smart Home Management
- 15+ Device Types: TV, AC, Fan, Light, Speaker, Camera, etc.
- 400+ Device Brands: Samsung, LG, Sony, Philips, MI, etc.
- 30+ Room Types: Bedroom, Kitchen, Living Room, etc.

⚡ Energy Management
- Real-time energy consumption tracking
- Indian electricity tariff calculations (6-slab system)
- Monthly cost analysis and efficiency recommendations

⏰ Automation Features
- Background timer service with 10-second monitoring
- Device scheduling with countdown displays
- Smart scenes for multiple device control

👥 Group Collaboration
- Multi-user device sharing
- Permission-based access control
- Group administration features

System Monitoring

DynamoDB Local Status
Health check
curl http://localhost:8002/

List tables (if any exist)
aws dynamodb list-tables --endpoint-url http://localhost:8002

Application Logs

- Console output shows real-time system events
- Security events with timestamps
- Timer execution notifications
- Energy consumption updates

System Architecture Overview

- Runtime: Java 21 LTS (Latest Long-Term Support)
- Database: Amazon DynamoDB (Local & AWS)
- Security: BCrypt password hashing + enhanced lockout system
- Build: Apache Maven 3.9.11
- Testing: JUnit 5 + Mockito
- Architecture: Clean MVC with Service Layer

Table of Contents

CustomerService.java - Enhanced Security & User Management {customer-service}

```
Package Declaration and Imports (Lines 1-18)
java
package com.smarthome.service;                  // Service layer package declaration

import com.smarthome.model.Customer;            // Customer entity for user data
import com.smarthome.util.DynamoDBConfig;       // Database configuration utility
import org.mindrot.jbcrypt.BCrypt;              // BCrypt for secure password hashing
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbEnhancedClient;
// Enhanced DynamoDB client for object mapping
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbTable;
// DynamoDB table interface for operations
import software.amazon.awssdk.enhanced.dynamodb.Key;
// DynamoDB key builder for queries
import software.amazon.awssdk.enhanced.dynamodb.TableSchema;
// Schema definition for entity mapping
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
// Exception for non-existent table scenarios

import java.time.LocalDateTime;                 // Date/time handling for security
logging
import java.util.ArrayList;                     // Dynamic array implementation
import java.util.Arrays;                        // Array utilities for common
passwords
import java.util.HashMap;                       // Hash map for demo mode storage
import java.util.List;                          // List interface for collections
```

```
import java.util.Map;                           // Map interface for key-value storage


 Class Declaration and Security Constants (Lines 19-34)
java
public class CustomerService {                 // User management service class

    private final DynamoDbTable<Customer> customerTable;  // DynamoDB table reference
    private final boolean isDemoMode;                      // Demo mode flag for
development
    private final Map<String, Customer> demoCustomers;    // In-memory storage for
demo mode

    // ENHANCED SECURITY: Comprehensive list of common passwords to reject
    private static final List<String> COMMON_PASSWORDS = Arrays.asList(
        "password", "123456", "password123", "admin", "qwerty", "abc123",
        // Line 26: Basic common passwords that attackers commonly try
        "123456789", "welcome", "monkey", "1234567890", "dragon", "letmein",
        // Line 27: Sequential and dictionary-based common passwords
        "india123", "mumbai", "delhi", "bangalore", "chennai", "kolkata",
        // Line 28: India-specific common passwords for regional security
        "password1", "admin123", "root", "toor", "pass", "test", "guest",
        // Line 29: Administrative and testing common passwords
        "user", "demo", "sample", "temp", "change", "changeme", "default",
        // Line 30: Default and temporary passwords often left unchanged
        "india", "bharat", "hindustan", "cricket", "bollywood", "iloveyou"
        // Line 31: Cultural and emotional passwords commonly used in India
    );
    // Total: 35+ common passwords blocked for enhanced security


 Constructor with Database Initialization (Lines 35-49)
java
public CustomerService() {                      // Default constructor
    DynamoDbEnhancedClient enhancedClient = DynamoDBConfig.getEnhancedClient();
    // Line 36: Gets enhanced DynamoDB client from configuration utility

    if (enhancedClient != null) {               // If database connection successful
        this.customerTable = enhancedClient.table("customers",
TableSchema.fromBean(Customer.class));
        // Line 39: Creates table reference with Customer entity schema mapping
        this.isDemoMode = false;                // Sets production database mode
        this.demoCustomers = null;              // No demo storage needed
        createTableIfNotExists();               // Ensures database table exists
        // Line 42: Calls method to create table if it doesn't exist
    } else {                                    // If database connection failed
        System.out.println("🎮 Running in DEMO MODE - data won't persist between
sessions");
        // Line 44: Warns user that data is temporary in demo mode
        this.customerTable = null;              // No database table available
        this.isDemoMode = true;                 // Enables demo mode flag
        this.demoCustomers = new HashMap<>();   // Initializes in-memory storage
        // Line 47: HashMap provides fast key-value lookup for demo users
    }
}


 Enhanced Authentication with Security Features (Lines 92-132)
java
public Customer authenticateCustomer(String email, String password) {
    try {
        email = email.trim().toLowerCase();     // Normalizes email format
        Customer customer = findCustomerByEmail(email); // Searches for user
```

```
        if (customer == null) {                 // ENHANCED: Specific error for
unregistered email
            System.out.println("[ERROR] Email not registered. Please create an account
first.");
            // Line 98: Clear message guides user to register instead of guessing
            // Add rate limiting even for non-existent accounts to prevent enumeration
            addSecurityDelay(1);                // Prevents rapid email enumeration
attacks
            // Line 100: 1-second delay makes bulk email scanning impractical
            return null;
        }

        if (customer.isAccountLocked()) {       // ENHANCED: Check account lockout
status
            LocalDateTime lockUntil = customer.getAccountLockedUntil();
            System.out.println("[LOCKED] Account is locked until: " +
                lockUntil.toString().replace("T", " ") +
                ". Please try again later.");
            // Lines 106-108: Shows exact unlock time for user convenience
            return null;                        // Prevents login during lockout
period
        }

        if (password != null && BCrypt.checkpw(password, customer.getPassword())) {
            // Line 112: BCrypt securely compares provided password with stored hash
            if (customer.getFailedLoginAttempts() > 0) { // Has previous failed
attempts
                customer.resetFailedAttempts(); // Clears failed attempt counter
                updateCustomer(customer);       // Persists reset to database
                System.out.println("[SUCCESS] Login successful! Previous failed
attempts have been cleared.");
                // Line 116: Informs user that security lockout has been reset
            } else {                            // Clean login with no previous
failures
                System.out.println("[SUCCESS] Login successful! Welcome back, " +
customer.getFullName() + "!");
            }
            return customer;                    // Returns authenticated user object
        } else {                               // Password verification failed
            // ENHANCED: Add progressive delay based on failed attempts before
handling failed login
            addSecurityDelay(customer.getFailedLoginAttempts() + 1);
            // Line 123: Delay increases with each failed attempt to slow brute force
            handleFailedLogin(customer);       // Processes failed login security
            return null;                       // Returns null for failed
authentication
        }

    } catch (Exception e) {                    // Handles unexpected authentication
errors
        System.err.println("Error authenticating customer: " + e.getMessage());
        return null;
    }
}


 Enhanced Lockout Policy (Lines 237-251)
java
private int calculateLockoutMinutes(int failedAttempts) {
    // ENHANCED SECURITY: More aggressive lockout policy than before
    if (failedAttempts >= 10) {
        return 1440; // 24 hours - severe security violation
```

```
        // Line 240: 10+ attempts = full day lockout for persistent attackers
    } else if (failedAttempts >= 7) {
        return 240;  // 4 hours - repeated violations
        // Line 242: 7-9 attempts = quarter day lockout for serious threats
    } else if (failedAttempts >= 5) {
        return 60;   // 1 hour - persistent attempts
        // Line 244: 5-6 attempts = hour lockout for sustained attacks
    } else if (failedAttempts >= 3) {
        return 15;   // 15 minutes - suspicious activity
        // Line 246: 3-4 attempts = quarter hour for moderate concern
    } else if (failedAttempts >= 2) {  // NEW: Earlier lockout at 2 attempts
        return 5;    // 5 minutes - first warning
        // Line 248: 2 attempts = 5 minute warning lockout (was 3 before)
    }
    return 0;        // 1 attempt = no lockout, just warning
}


 Enhanced Failed Login Handler (Lines 253-282)
java
private void handleFailedLogin(Customer customer) {
    customer.incrementFailedAttempts();         // Increases failed attempt counter
    int attempts = customer.getFailedLoginAttempts(); // Gets current attempt count

    // ENHANCED: Security logging with timestamp and email tracking
    System.out.println("[SECURITY] Failed login attempt " + attempts + " for account:
" +
                       customer.getEmail() + " at " +
java.time.LocalDateTime.now().toString().replace("T", " "));
    // Lines 258-259: Detailed logging for security monitoring and forensics

    int lockoutMinutes = calculateLockoutMinutes(attempts); // Calculates lockout
duration
    if (lockoutMinutes > 0) {                   // Lockout required
        customer.lockAccount(lockoutMinutes);   // Sets lockout timestamp

        if (lockoutMinutes >= 1440) {           // 24-hour lockout (severe)
            System.out.println("[SECURITY ALERT] Account PERMANENTLY LOCKED for 24
hours due to " +
                               attempts + " failed attempts. Contact administrator if
legitimate.");
            // Lines 266-267: Highest severity message for persistent attacks
        } else if (lockoutMinutes >= 240) {     // 4-hour lockout (high risk)
            System.out.println("[HIGH SECURITY RISK] Account locked for " +
(lockoutMinutes/60) + " hours due to " +
                               attempts + " repeated failed attempts.");
            // Lines 269-270: High severity message for serious threats
        } else {                                // Standard lockout (15 minutes or 1
hour)
            System.out.println("[LOCKED] Account locked for " + lockoutMinutes + "
minutes due to " +
                               attempts + " failed login attempts.");
            // Lines 272-273: Standard lockout message with duration
        }
    } else {                                    // First attempt - no lockout yet
        // ENHANCED: Clear warning about upcoming lockout after just 2 attempts
        System.out.println("[WARNING] Invalid password. This is attempt " + attempts +
" of 2 allowed before lockout.");
        System.out.println("[SECURITY] Account will be locked after 2 failed attempts
for security.");
        // Lines 277-278: Early warning system informs users of strict policy
    }
```

```
    updateCustomer(customer);                   // Persists security changes to
database
}


 Rate Limiting Security Method (Lines 431-462)
java
/
  ENHANCED SECURITY: Progressive delay to prevent rapid brute force attacks
  Delays increase exponentially with failed attempts to slow down attackers
 /
private void addSecurityDelay(int failedAttempts) {
    try {
        int delaySeconds;                       // Variable to store delay duration
        switch (failedAttempts) {               // Progressive delay based on attempt
number
            case 1:
                delaySeconds = 1;  // 1 second - first failed attempt
                // Line 436: Minimal delay for first failure (could be typo)
                break;
            case 2:
                delaySeconds = 3;  // 3 seconds - second attempt
                // Line 439: Moderate delay suggests possible attack
                break;
            case 3:
                delaySeconds = 5;  // 5 seconds - getting suspicious
                // Line 442: Longer delay for sustained attempts
                break;
            case 4:
                delaySeconds = 10; // 10 seconds - definite attack pattern
                // Line 445: Significant delay for clear attack behavior
                break;
            default:
                delaySeconds = 15; // 15 seconds - severe attack pattern
                // Line 448: Maximum delay for persistent attackers
                break;
        }

        if (delaySeconds > 1) {                 // Only announce longer delays
            System.out.println("[SECURITY] Implementing " + delaySeconds + " second
delay due to failed attempts...");
            // Line 453: Informs attacker that delays are active (deterrent effect)
        }

        Thread.sleep(delaySeconds  1000L);   // Implements the actual delay
        // Line 456: Blocks current thread for calculated duration

    } catch (InterruptedException e) {          // Handle thread interruption
        Thread.currentThread().interrupt();   // Restore interrupt status
        System.err.println("[SECURITY] Security delay interrupted");
        // Line 460: Logs if delay is interrupted (should not happen normally)
    }
}
```

SmartHomeDashboard.java - Main Application with Enhanced Input Handling {main-app}

```
Package Declaration and Imports (Lines 1-11)
java
package com.smarthome;                          // Root package declaration

import com.smarthome.model.Customer;            // Customer entity for user operations
import com.smarthome.model.Gadget;              // Gadget entity for device operations
```

```
import com.smarthome.service.SmartHomeService;  // Main service orchestrator
import com.smarthome.util.DynamoDBConfig;       // Database configuration utility

import java.util.ArrayList;                     // Dynamic list for collections
import java.util.List;                          // List interface for device
collections
import java.util.Scanner;                       // CONSISTENT input handling
throughout app


 Class Declaration and Constants (Lines 12-127)
java
public class SmartHomeDashboard {               // Main application class - entry
point

    private static final Scanner scanner = new Scanner(System.in);
    // Line 14: CRITICAL - Single Scanner instance for consistent input handling
    // This prevents buffer conflicts that caused double-enter issues

    private static final SmartHomeService smartHomeService = new SmartHomeService();
    // Line 15: Main service instance - handles all business logic and orchestration

    private static volatile boolean returnToMainMenu = false;
    // Line 16: Thread-safe flag for navigation control (volatile ensures visibility)

    // COMPREHENSIVE DEVICE BRAND ARRAYS for user convenience
    private static final String[] TV_BRANDS = {
        "Samsung", "Sony", "LG", "TCL", "Hisense", "Panasonic", "Philips", "MI",
"OnePlus",
        "Xiaomi", "Realme", "Redmi", "Vu", "Thomson", "Kodak", "Motorola", "Nokia",
        "Toshiba", "Sharp", "Haier", "BPL", "Videocon", "Intex", "Micromax", "Shinco"
    };
    // Lines 18-22: 25 TV brands covering global and Indian market preferences

    private static final String[] AC_BRANDS = {
        "LG", "Voltas", "Blue Star", "Samsung", "Daikin", "Hitachi", "Panasonic",
"Carrier",
        "Godrej", "Haier", "Whirlpool", "Lloyd", "O General", "Mitsubishi", "Toshiba",
        "Electrolux", "IFB", "Crompton", "Orient", "Bajaj", "Usha", "Havells",
"Symphony"
    };
    // Lines 24-28: 23 AC brands including premium and Indian manufacturers

    // Similar comprehensive arrays continue for:
    // FAN_BRANDS (22 brands), SPEAKER_BRANDS (21 brands), AIR_PURIFIER_BRANDS (21
brands)
    // THERMOSTAT_BRANDS (20 brands), LIGHT_BRANDS (22 brands), SWITCH_BRANDS (20
brands)
    // CAMERA_BRANDS (24 brands), LOCK_BRANDS (22 brands), DOORBELL_BRANDS (22 brands)
    // REFRIGERATOR_BRANDS (22 brands), MICROWAVE_BRANDS (22 brands),
WASHING_MACHINE_BRANDS (22 brands)
    // GEYSER_BRANDS (22 brands), WATER_PURIFIER_BRANDS (20 brands), VACUUM_BRANDS (22
brands)
    // ROOM_NAMES (30 room types)
    // Total: 400+ brand options + 30 room types for comprehensive device support


 Enhanced Main Method with Graceful Shutdown (Lines 128-160)
java
public static void main(String[] args) {        // Application entry point
    System.out.println("=== Welcome to IoT Smart Home Dashboard ===\n");
    // Line 129: Updated title (removed "Enterprise" per requirements)
```

```
    System.out.println("🔍 Testing DynamoDB connection...");
    DynamoDBConfig.testConnection();             // Tests database connectivity
    System.out.println();                        // Blank line for readability

    Runtime.getRuntime().addShutdownHook(new Thread(() -> {
        // Lines 135-147: ENHANCED graceful shutdown handler
        System.out.println("\n[SYSTEM] Graceful shutdown initiated...");
        try {
            if (smartHomeService.isLoggedIn()) { // Checks if user is logged in
                smartHomeService.logout();       // Properly logout user
                System.out.println("[SYSTEM] User session closed.");
            }
            smartHomeService.getTimerService().shutdown(); // Stops background timers
            System.out.println("[SYSTEM] Timer service shutdown completed.");
        } catch (Exception e) {                  // Handles shutdown errors gracefully
            System.err.println("[SYSTEM] Warning during shutdown: " + e.getMessage());
        }
    }));

    displayNavigationHelp();                     // Shows navigation instructions

    try {
        showMainMenu();                          // Starts main application loop
    } catch (Exception e) {                      // Global exception handler
        System.err.println("Application error: " + e.getMessage());
        e.printStackTrace();                     // Debug information for developers
    } finally {                                  // Cleanup regardless of how app exits
        DynamoDBConfig.shutdown();               // Closes database connections
        scanner.close();                         // Closes input scanner
    }
}


 CRITICAL: Enhanced Password Input Method (Lines 3343-3352)
java
/
  PERMANENT SOLUTION: Reads password using consistent scanner input to prevent buffer
conflicts
  Note: Password will be visible but ensures single-enter navigation works properly
 /
private static String readMaskedPassword() throws Exception {
    // DECISION: Chose functionality over password masking for stability
    // Previous attempts with console.readPassword() caused double-enter issues

    // Use only scanner.nextLine() for consistent input handling
    // This eliminates buffer conflicts that cause double-enter issues
    String password = scanner.nextLine();       // CONSISTENT with all other input
    return password != null ? password.trim() : ""; // Returns cleaned password
}
// Lines 3347-3352: PERMANENT FIX for navigation issues
// Trade-off: Visible passwords but reliable single-enter navigation
// Security maintained through aggressive lockout policies instead
```

Customer.java - User Model with Enhanced Security Features {customer-model}

```
Package Declaration and DynamoDB Imports (Lines 1-10)
java
package com.smarthome.model;                    // Model package for entity classes

import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.DynamoDbBean;
// Line 3: DynamoDB enhanced annotation for entity mapping
import
software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.DynamoDbPartitionKey;
```

```
// Line 4: Annotation to mark partition key field for DynamoDB table structure
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.DynamoDbAttribute;
// Line 5: Annotation to customize DynamoDB attribute mapping

import java.time.LocalDateTime;                // Modern Java date/time handling
import java.util.ArrayList;                    // Dynamic array for collections
import java.util.List;                         // List interface for type safety


 Entity Class Declaration with Security Fields (Lines 11-26)
java
@DynamoDbBean                                  // Marks class as DynamoDB entity
public class Customer {                        // Main user entity class

    // CORE USER FIELDS
    private String email;                      // Unique identifier (partition key)
    private String fullName;                   // User's display name
    private String password;                   // BCrypt hashed password (never plain
text)
    private List<Gadget> gadgets;              // User's connected smart devices
    private List<String> groupMembers;         // Email list of group collaborators
    private String groupCreator;               // Email of group administrator
    private List<DeletedDeviceEnergyRecord> deletedDeviceEnergyRecords;
    // Line 20: Preserves energy data for accurate monthly billing
    private List<DevicePermission> devicePermissions;
    // Line 21: Device access control for group members

    // ENHANCED SECURITY FIELDS (new implementation)
    private int failedLoginAttempts;           // Counter for progressive lockout
    private LocalDateTime accountLockedUntil; // Timestamp for lockout expiration
    private LocalDateTime lastFailedLoginTime; // Track attack patterns


 Default Constructor with Security Initialization (Lines 28-37)
java
public Customer() {                            // Default constructor for DynamoDB
    this.gadgets = new ArrayList<>();          // Initialize empty device list
    this.groupMembers = new ArrayList<>();     // Initialize empty group member list
    this.groupCreator = null;                  // No group admin initially
    this.deletedDeviceEnergyRecords = new ArrayList<>();
    // Line 32: Initialize energy history preservation list
    this.devicePermissions = new ArrayList<>();
    // Line 33: Initialize device permission control list
    this.failedLoginAttempts = 0;              // Start with clean security record
    this.accountLockedUntil = null;            // No lockout initially
    this.lastFailedLoginTime = null;           // No failed attempts initially
}


 Parameterized Constructor for Registration (Lines 39-51)
java
public Customer(String email, String fullName, String password) {
    this.email = email;                        // Set user's unique email identifier
    this.fullName = fullName;                  // Set user's display name
    this.password = password;                  // Set BCrypt hashed password
    this.gadgets = new ArrayList<>();          // Initialize empty device collection
    this.groupMembers = new ArrayList<>();     // Initialize empty group collection
    this.groupCreator = null;                  // Not part of group initially
    this.deletedDeviceEnergyRecords = new ArrayList<>();
    // Line 46: Energy preservation for accurate billing
    this.devicePermissions = new ArrayList<>();
    // Line 47: Device access control initialization
    this.failedLoginAttempts = 0;              // Clean security state
```

```
    this.accountLockedUntil = null;          // No security lockout
    this.lastFailedLoginTime = null;         // No attack history
}


 DynamoDB Partition Key and Basic Getters/Setters (Lines 53-84)
java
@DynamoDbPartitionKey                         // Marks email as primary key
public String getEmail() {                    // DynamoDB requires public getter
    return email;
}

public void setEmail(String email) {          // Setter for email updates
    this.email = email;
}

// Standard getters and setters for basic fields (Lines 62-84)
// getFullName(), setFullName(), getPassword(), setPassword()
// getGadgets(), setGadgets() with null safety check


 Smart Device Management Methods (Lines 86-110)
java
public void addGadget(Gadget gadget) {        // Add device with duplicate prevention
    if (this.gadgets == null) {               // Defensive null check
        this.gadgets = new ArrayList<>();     // Initialize if needed
    }

    boolean exists = this.gadgets.stream()    // Check for existing device
            .anyMatch(g -> g.getType().equalsIgnoreCase(gadget.getType()) &&
                           g.getRoomName().equalsIgnoreCase(gadget.getRoomName()));
    // Lines 91-93: Prevents duplicate devices of same type in same room

    if (!exists) {                            // Only add if unique
        this.gadgets.add(gadget);             // Add device to collection
    }
}

public Gadget findGadget(String type, String roomName) {
    if (this.gadgets == null) {               // Safety check for null collection
        return null;
    }

    return this.gadgets.stream()              // Stream-based search
            .filter(g -> g.getType().equalsIgnoreCase(type) &&
                         g.getRoomName().equalsIgnoreCase(roomName))
    // Lines 105-107: Case-insensitive matching for device lookup
            .findFirst()                      // Get first match
            .orElse(null);                    // Return null if not found
}


 Enhanced Security State Management (Lines 112-153)
java
// Security field getters/setters (Lines 112-134)
public int getFailedLoginAttempts() { return failedLoginAttempts; }
public void setFailedLoginAttempts(int failedLoginAttempts) {
    this.failedLoginAttempts = failedLoginAttempts;
}

public boolean isAccountLocked() {            // CRITICAL security method
    return accountLockedUntil != null &&
LocalDateTime.now().isBefore(accountLockedUntil);
```

```
    // Line 137: Real-time lockout status check using current timestamp
}

public void incrementFailedAttempts() {      // Called on each failed login
    this.failedLoginAttempts++;              // Increase failure counter
    this.lastFailedLoginTime = LocalDateTime.now();
    // Line 142: Record timestamp for attack pattern analysis
}

public void resetFailedAttempts() {          // Called on successful login
    this.failedLoginAttempts = 0;            // Clear failure counter
    this.accountLockedUntil = null;          // Remove lockout restriction
    this.lastFailedLoginTime = null;         // Clear attack history
}

public void lockAccount(int minutes) {       // Progressive lockout implementation
    this.accountLockedUntil = LocalDateTime.now().plusMinutes(minutes);
    // Line 152: Set lockout expiration based on failure count
}


 Group Collaboration Management (Lines 155-209)
java
// Group member management with normalization
public void addGroupMember(String memberEmail) {
    if (this.groupMembers == null) {         // Defensive initialization
        this.groupMembers = new ArrayList<>();
    }

    if (!this.groupMembers.contains(memberEmail.toLowerCase().trim())) {
        this.groupMembers.add(memberEmail.toLowerCase().trim());
        // Line 169: Normalize email format to prevent duplicates
    }
}

public void removeGroupMember(String memberEmail) {
    if (this.groupMembers != null) {
        this.groupMembers.remove(memberEmail.toLowerCase().trim());
        // Line 175: Case-insensitive removal with normalization
    }
}

// Group status and administration methods
public boolean isPartOfGroup() {             // Check if user belongs to any group
    return this.groupMembers != null && !this.groupMembers.isEmpty();
}

public boolean isGroupAdmin() {              // Check if user is group administrator
    return this.groupCreator != null &&
this.groupCreator.equalsIgnoreCase(this.email);
    // Line 196: Compare creator email with current user's email
}

public int getGroupSize() {                  // Calculate total group members
    int size = 0;
    if (this.groupMembers != null) {
        size += this.groupMembers.size();    // Count invited members
    }
    return size + 1;                         // +1 for the current user
}


 Energy History Preservation System (Lines 211-242)
```

```
java
@DynamoDbAttribute("deletedDeviceEnergyRecords")
public List<DeletedDeviceEnergyRecord> getDeletedDeviceEnergyRecords() {
    if (deletedDeviceEnergyRecords == null) {    // Lazy initialization
        deletedDeviceEnergyRecords = new ArrayList<>();
    }
    return deletedDeviceEnergyRecords;
}

public void addDeletedDeviceRecord(DeletedDeviceEnergyRecord record) {
    if (deletedDeviceEnergyRecords == null) {
        deletedDeviceEnergyRecords = new ArrayList<>();
    }
    deletedDeviceEnergyRecords.add(record);     // Preserve energy data
}

public double getTotalDeletedDeviceEnergyForCurrentMonth() {
    if (deletedDeviceEnergyRecords == null || deletedDeviceEnergyRecords.isEmpty()) {
        return 0.0;                             // No deleted devices
    }

    LocalDateTime now = LocalDateTime.now();
    String currentMonth = now.getYear() + "-" + String.format("%02d",
now.getMonthValue());
    // Line 236: Format current month as "YYYY-MM" for filtering

    return deletedDeviceEnergyRecords.stream()  // Stream processing
            .filter(record -> currentMonth.equals(record.getDeletionMonth()))
    // Line 239: Filter records for current month only
            .mapToDouble(DeletedDeviceEnergyRecord::getTotalEnergyConsumedKWh)
    // Line 240: Extract energy consumption values
            .sum();                             // Sum all energy from deleted devices
    // This method ensures accurate monthly billing despite device deletions
}


 Device Permission Control System (Lines 244-331)
java
@DynamoDbAttribute("devicePermissions")
public List<DevicePermission> getDevicePermissions() {
    if (devicePermissions == null) {           // Lazy initialization pattern
        devicePermissions = new ArrayList<>();
    }
    return devicePermissions;
}

public boolean grantDevicePermission(String memberEmail, String deviceType, String
roomName, String grantedBy) {
    // Check if device exists
    Gadget device = findGadget(deviceType, roomName);
    if (device == null) {                      // Device must exist to grant
permission
        return false;
    }

    // Check if permission already exists
    if (hasDevicePermission(memberEmail, deviceType, roomName)) {
        return false;                          // Prevent duplicate permissions
    }

    // Create new permission
    DevicePermission permission = new DevicePermission(memberEmail, deviceType,
roomName, this.email, grantedBy);
```

```
    getDevicePermissions().add(permission);     // Add to permission list
    return true;                                // Successfully granted
}

public boolean revokeDevicePermission(String memberEmail, String deviceType, String
roomName) {
    return getDevicePermissions().removeIf(permission ->
        permission.getMemberEmail().equalsIgnoreCase(memberEmail) &&
        permission.matchesDevice(deviceType, roomName, this.email));
    // Lines 281-283: Remove permission if matching member and device found
}

public boolean hasDevicePermission(String memberEmail, String deviceType, String
roomName) {
    return getDevicePermissions().stream()      // Stream-based permission check
        .anyMatch(permission ->
            permission.getMemberEmail().equalsIgnoreCase(memberEmail) &&
            permission.matchesDevice(deviceType, roomName, this.email));
    // Lines 291-293: Case-insensitive permission verification
}

public List<Gadget> getAccessibleGroupDevices(List<Customer> groupMembers) {
    List<Gadget> accessibleDevices = new ArrayList<>();

    for (Customer member : groupMembers) {
        if (member.getEmail().equalsIgnoreCase(this.email)) {
            continue;                           // Skip self
        }

        // Check what permissions this member has granted to current user
        List<DevicePermission> permissionsForMe =
member.getPermissionsForMember(this.email);

        for (DevicePermission permission : permissionsForMe) {
            if (permission.isCanView()) {       // Check if view permission granted
                Gadget device = member.findGadget(permission.getDeviceType(),
permission.getRoomName());
                if (device != null) {
                    accessibleDevices.add(device);  // Add accessible device
                }
            }
        }
    }

    return accessibleDevices;                   // Return all accessible devices
}


 Comprehensive ToString Method (Lines 333-342)
java
@Override
public String toString() {                      // Debug-friendly string representation
    return "Customer{" +
            "email='" + email + '\'' +          // Show unique identifier
            ", fullName='" + fullName + '\'' + // Show display name
            ", gadgetsCount=" + (gadgets != null ? gadgets.size() : 0) +
            // Line 338: Safe gadget count even if null
            ", failedAttempts=" + failedLoginAttempts +
            // Line 339: Show security status
            ", isLocked=" + isAccountLocked() + // Real-time lockout status
            '}';
}
```

Gadget.java - Device Model with Energy Tracking {gadget-model}

```
Package Declaration and Imports (Lines 1-6)
java
package com.smarthome.model;                    // Model package for device entities

import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.DynamoDbBean;
// Line 3: DynamoDB enhanced annotation for seamless object mapping
import java.time.LocalDateTime;                 // Modern Java 8+ date/time API
import java.time.temporal.ChronoUnit;           // Time calculation utilities for
energy tracking


 Device Type Enumerations (Lines 7-16)
java
@DynamoDbBean                                   // Marks class as DynamoDB entity
public class Gadget {

    public enum GadgetType {                    // Device type constants
        TV, AC, FAN, ROBO_VAC_MOP              // Enumerated device types for type
safety
        // Line 11: Enum prevents invalid device type assignments
    }

    public enum GadgetStatus {                  // Device state constants
        ON, OFF                                // Binary status for device power state
        // Line 15: Enum ensures consistent status representation
    }


 Core Device Properties (Lines 18-29)
java
// DEVICE IDENTIFICATION
private String type;                           // Device category (TV, AC, FAN, etc.)
private String model;                          // Brand/model name for user
identification
private String roomName;                       // Location within home for organization
private String status;                         // Current power state (ON/OFF as
string)

// ENERGY MANAGEMENT FIELDS
private double powerRatingWatts;               // Device power consumption in watts
private LocalDateTime lastOnTime;              // Timestamp when device was last turned
ON
private LocalDateTime lastOffTime;             // Timestamp when device was last turned
OFF
private long totalUsageMinutes;                // Cumulative usage time in minutes
private double totalEnergyConsumedKWh;         // Total energy consumed in kilowatt-
hours

// TIMER AUTOMATION FIELDS
private LocalDateTime scheduledOnTime;         // Future timestamp for automatic ON
private LocalDateTime scheduledOffTime;        // Future timestamp for automatic OFF
private boolean timerEnabled;                  // Flag indicating if timer is active


 Default Constructor with Energy Initialization (Lines 31-37)
java
public Gadget() {                              // Default constructor for DynamoDB
    this.status = GadgetStatus.OFF.name();     // Start in OFF state for safety
    this.powerRatingWatts = 0.0;               // No power consumption initially
    this.totalUsageMinutes = 0L;               // No usage history
    this.totalEnergyConsumedKWh = 0.0;         // No energy consumed yet
```

```
    this.timerEnabled = false;                // No timers active by default
}


 Parameterized Constructor with Auto Power Rating (Lines 39-48)
java
public Gadget(String type, String model, String roomName) {
    this.type = type;                         // Set device category
    this.model = model;                       // Set brand/model identification
    this.roomName = roomName;                 // Set room location
    this.status = GadgetStatus.OFF.name();    // Default to OFF state
    this.powerRatingWatts = getDefaultPowerRating(type);
    // Line 44: Automatically assign power rating based on device type
    this.totalUsageMinutes = 0L;              // Initialize usage tracking
    this.totalEnergyConsumedKWh = 0.0;        // Initialize energy tracking
    this.timerEnabled = false;                // No automation initially
}


 Smart Power Rating Assignment with Type Checking (Lines 54-59)
java
public void setType(String type) {            // Setter with intelligent power rating
    this.type = type;                         // Set device type
    if (this.powerRatingWatts == 0.0) {       // If no power rating set
        this.powerRatingWatts = getDefaultPowerRating(type);
        // Line 57: Automatically assign appropriate power rating
    }
}


 Device Status Control with Energy Tracking (Lines 85-111)
java
public boolean isOn() {                       // Check if device is currently running
    return GadgetStatus.ON.name().equals(this.status);
    // Line 86: String comparison for reliable status checking
}

public void turnOn() {                        // Turn device ON with energy tracking
    if (!isOn()) {                            // Only act if currently OFF
        this.lastOnTime = LocalDateTime.now(); // Record ON timestamp
        updateUsageAndEnergy();               // Update energy consumption
        // Line 92: Calculate energy for previous session before starting new one
    }
    this.status = GadgetStatus.ON.name();     // Set status to ON
}

public void turnOff() {                       // Turn device OFF with energy tracking
    if (isOn()) {                             // Only act if currently ON
        this.lastOffTime = LocalDateTime.now(); // Record OFF timestamp
        updateUsageAndEnergy();               // Calculate energy for this session
        // Line 100: Critical energy calculation before turning OFF
    }
    this.status = GadgetStatus.OFF.name();    // Set status to OFF
}

public void toggleStatus() {                  // Smart toggle with energy tracking
    if (isOn()) {                             // If currently ON
        turnOff();                            // Turn OFF (with energy calculation)
    } else {                                  // If currently OFF
        turnOn();                             // Turn ON (with timestamp recording)
    }
}
```

```
 CRITICAL: Real-Time Energy Calculation System (Lines 113-122)
java
private void updateUsageAndEnergy() {         // Core energy tracking method
    if (lastOnTime != null && isOn()) {      // Only calculate if device was ON
        long minutesUsed = ChronoUnit.MINUTES.between(lastOnTime,
LocalDateTime.now());
        // Line 115: Calculate exact minutes device was running
        totalUsageMinutes += minutesUsed;    // Add to cumulative usage time

        double hoursUsed = minutesUsed / 60.0; // Convert minutes to hours
        double energyUsed = (powerRatingWatts / 1000.0)  hoursUsed;
        // Line 119: Energy = Power (kW) × Time (hours) = kWh
        totalEnergyConsumedKWh += energyUsed; // Add to total energy consumption
        // This method is called every time device status changes
    }
}


 Comprehensive Default Power Ratings Database (Lines 124-145)
java
private static double getDefaultPowerRating(String deviceType) {
    switch (deviceType.toUpperCase()) {      // Case-insensitive device matching
        case "TV": return 150.0;             // LED/LCD TV average consumption
        case "AC": return 1500.0;            // Split AC 1.5 ton average
        case "FAN": return 75.0;             // Ceiling fan average
        case "LIGHT": return 60.0;           // LED bulb equivalent
        case "SPEAKER": return 30.0;         // Smart speaker average
        case "AIR_PURIFIER": return 45.0;    // Home air purifier average
        case "THERMOSTAT": return 5.0;       // Smart thermostat minimal power
        case "SWITCH": return 2.0;           // Smart switch standby power
        case "CAMERA": return 15.0;          // Security camera average
        case "DOOR_LOCK": return 12.0;       // Smart lock average
        case "DOORBELL": return 8.0;         // Video doorbell average
        case "REFRIGERATOR": return 200.0;   // Standard refrigerator
        case "MICROWAVE": return 1200.0;     // Microwave oven high power
        case "WASHING_MACHINE": return 500.0; // Front-load washer average
        case "GEYSER": return 2000.0;        // Electric water heater maximum
        case "WATER_PURIFIER": return 25.0; // RO water purifier average
        case "VACUUM": return 1400.0;        // Vacuum cleaner high power
        default: return 50.0;                // Generic smart device fallback
        // Lines 126-144: Comprehensive power database for 16+ device types
    }
}


 Real-Time Energy Calculation Methods (Lines 217-248)
java
public double getCurrentSessionUsageHours() { // Calculate current session usage
    if (isOn() && lastOnTime != null) {      // Only if device is currently ON
        long currentMinutes = ChronoUnit.MINUTES.between(lastOnTime,
LocalDateTime.now());
        // Line 219: Real-time minutes since device was turned ON
        return currentMinutes / 60.0;        // Convert to hours for energy calculation
    }
    return 0.0;                              // No current session if OFF
}

public double getCurrentTotalEnergyConsumedKWh() {
    double baseEnergy = totalEnergyConsumedKWh; // Start with historical energy
    if (isOn() && lastOnTime != null) {         // Add current session if running
        double currentSessionHours = getCurrentSessionUsageHours();
```

```
        double currentSessionEnergy = (powerRatingWatts / 1000.0)
currentSessionHours;
        // Line 229: Calculate energy for current session in real-time
        baseEnergy += currentSessionEnergy;     // Add to historical total
    }
    return baseEnergy;                          // Return complete energy consumption
    // This method provides real-time energy without modifying stored values
}

public long getCurrentTotalUsageMinutes() {    // Real-time usage time calculation
    long baseUsage = totalUsageMinutes;        // Start with historical usage
    if (isOn() && lastOnTime != null) {        // Add current session if running
        long currentMinutes = ChronoUnit.MINUTES.between(lastOnTime,
LocalDateTime.now());
        baseUsage += currentMinutes;           // Add current session minutes
    }
    return baseUsage;                          // Return total usage including current
session
}

public String getCurrentUsageTimeFormatted() { // Human-readable usage time
    long totalCurrentMinutes = getCurrentTotalUsageMinutes();
    long hours = totalCurrentMinutes / 60;     // Extract hours
    long minutes = totalCurrentMinutes % 60;   // Extract remaining minutes
    return String.format("%dh %02dm", hours, minutes);
    // Line 248: Format as "123h 45m" for user display
}


 Device Identification and Comparison (Lines 251-274)
java
@Override
public String toString() {                     // User-friendly device description
    return String.format("%s %s in %s - %s (%.1fW)", type, model, roomName, status,
powerRatingWatts);
    // Line 253: Complete device info in single string format
}

public void ensurePowerRating() {              // Power rating validation method
    if (this.powerRatingWatts == 0.0 && this.type != null) {
        this.powerRatingWatts = getDefaultPowerRating(this.type);
        // Line 258: Auto-fix missing power ratings for existing devices
    }
}

@Override
public boolean equals(Object obj) {            // Device equality based on type and
location
    if (this == obj) return true;             // Same object reference
    if (obj == null || getClass() != obj.getClass()) return false;

    Gadget gadget = (Gadget) obj;
    return type.equals(gadget.type) && roomName.equals(gadget.roomName);
    // Line 268: Devices are equal if same type in same room (prevents duplicates)
}

@Override
public int hashCode() {                        // Hash code for efficient collections
    return (type + roomName).hashCode();       // Combine type and room for unique hash
    // Line 273: Enables efficient HashMap/HashSet operations
}
```

SmartHomeService.java - Main Service Orchestrator {main-service}

```
Package Declaration and Service Integration Imports (Lines 1-15)
java
package com.smarthome.service;                // Service layer package

import com.smarthome.model.Customer;          // User entity for operations
import com.smarthome.model.Gadget;            // Device entity for management
import com.smarthome.model.DeletedDeviceEnergyRecord; // Energy preservation
import com.smarthome.util.SessionManager;     // Session state management

import java.util.ArrayList;                   // Dynamic collections
import java.util.HashMap;                     // Key-value mappings
import java.util.List;                        // List interface
import java.util.Map;                         // Map interface
import java.time.LocalDateTime;               // Date/time operations
import java.time.format.DateTimeFormatter;   // Time formatting
import java.time.temporal.ChronoUnit;         // Time calculations


 Service Architecture and Dependency Injection (Lines 16-38)
java
public class SmartHomeService {               // Central service orchestrator

    // CORE SERVICE DEPENDENCIES - Composition pattern for modularity
    private final CustomerService customerService;        // User management
    private final GadgetService gadgetService;           // Device management
    private final SessionManager sessionManager;         // Session state
    private final EnergyManagementService energyService; // Energy calculations
    private final TimerService timerService;            // Background automation
    private final CalendarEventService calendarService; // Event scheduling
    private final WeatherService weatherService;        // Weather integration
    private final SmartScenesService smartScenesService; // Scene automation
    private final DeviceHealthService deviceHealthService; // Health monitoring

    public SmartHomeService() {               // Constructor with service
initialization
        this.customerService = new CustomerService();    // User management service
        this.gadgetService = new GadgetService();        // Device management service
        this.sessionManager = SessionManager.getInstance(); // Singleton session
manager
        this.energyService = new EnergyManagementService(); // Energy calculation
service
        this.timerService = TimerService.getInstance(customerService); // Singleton
timer service
        this.calendarService = CalendarEventService.getInstance(); // Singleton
calendar service
        this.weatherService = WeatherService.getInstance(); // Singleton weather
service
        this.smartScenesService = SmartScenesService.getInstance(); // Singleton
scenes service
        this.deviceHealthService = DeviceHealthService.getInstance(); // Singleton
health service
        // Lines 28-37: Singleton pattern ensures single instance of shared services
    }


 Enhanced User Registration with Comprehensive Validation (Lines 44-74)
java
public boolean registerCustomer(String fullName, String email, String password, String
confirmPassword) {
    if (!password.equals(confirmPassword)) {  // Password confirmation check
        System.out.println("Passwords do not match!");
```

```
        return false;
    }

    if (!customerService.isValidName(fullName)) { // Name format validation
        System.out.println("Invalid name! Name should contain only letters and spaces
(minimum 2 characters).");
        return false;
    }

    if (!customerService.isValidEmail(email)) { // Email format validation
        System.out.println("Invalid email format!");
        return false;
    }

    if (!customerService.isValidPassword(password)) { // Enhanced password validation
        System.out.println("[ERROR] Invalid password! Please ensure your password
meets all requirements:");
        System.out.println(customerService.getPasswordRequirements());
        // Lines 61-62: Display comprehensive password requirements
        return false;
    }

    boolean success = customerService.registerCustomer(fullName, email, password);
    if (success) {
        System.out.println("[SUCCESS] Thank you! Customer registration successful.");
    } else {
        System.out.println("[ERROR] Registration failed! Email might already be
registered.");
    }

    return success;                          // Return registration result
}


 Secure Authentication with Session Management (Lines 76-93)
java
public boolean loginCustomer(String email, String password) {
    Customer customer = customerService.authenticateCustomer(email, password);
    // Line 77: Delegates to CustomerService for enhanced security checks

    if (customer != null) {                  // Authentication successful
        sessionManager.login(customer);     // Start user session
        return true;                         // Login successful
    } else {                                 // Authentication failed
        return false;                        // Security handled by CustomerService
    }
}

public void logout() {                       // Secure logout with session cleanup
    String currentUserEmail = sessionManager.getCurrentUser() != null ?
                              sessionManager.getCurrentUser().getEmail() : "unknown";
    // Lines 88-89: Capture email before session cleanup
    sessionManager.logout();                // Clear session state
    System.out.println("[SUCCESS] Logged out successfully from account: " +
currentUserEmail);
    System.out.println("[INFO] You can now register a new account or login with
different credentials.");
}


 Smart Device Connection with Duplicate Prevention (Lines 95-132)
java
public boolean connectToGadget(String type, String model, String roomName) {
```

```
    if (!sessionManager.isLoggedIn()) {     // Session validation
        System.out.println("Please login first!");
        return false;
    }

    try {
        Customer currentUser = sessionManager.getCurrentUser();

        Gadget existingGadget = currentUser.findGadget(type, roomName);
        if (existingGadget != null) {       // Prevent duplicate devices
            System.out.println("A " + type + " already exists in " + roomName +
                               ". You can only have one " + type + " per room.");
            return false;
        }

        Gadget gadget = gadgetService.createGadget(type, model, roomName);
        gadget.ensurePowerRating();         // Ensure power rating is set
        currentUser.addGadget(gadget);      // Add to user's device collection

        boolean updated = customerService.updateCustomer(currentUser);

        if (updated) {                      // Database update successful
            sessionManager.updateCurrentUser(currentUser); // Sync session
            System.out.println("[SUCCESS] Successfully connected to " +
                               gadget.getType() + " " + gadget.getModel() +
                               " in " + gadget.getRoomName());
            return true;
        } else {
            System.out.println("[ERROR] Failed to update customer data!");
            return false;
        }

    } catch (IllegalArgumentException e) {   // Handle invalid arguments
        System.out.println("Error: " + e.getMessage());
        return false;
    } catch (Exception e) {                 // Handle unexpected errors
        System.out.println("Unexpected error occurred. Please try again.");
        return false;
    }
}


 Advanced Device Viewing with Group Integration (Lines 134-190)
java
public List<Gadget> viewGadgets() {         // View devices with group support
    if (!sessionManager.isLoggedIn()) {     // Authentication check
        System.out.println("Please login first!");
        return null;
    }

    Customer currentUser = sessionManager.getCurrentUser();
    List<Gadget> allGadgets = new ArrayList<>();

    if (currentUser.isPartOfGroup()) {      // GROUP MODE: Show personal + shared
devices
        if (currentUser.getGadgets() != null) {
            allGadgets.addAll(currentUser.getGadgets()); // Add personal devices
        }

        int groupDeviceCount = 0;
        List<Customer> groupMemberObjects = new ArrayList<>();

        // Fetch group member objects
```

```
        for (String memberEmail : currentUser.getGroupMembers()) {
            Customer groupMember = customerService.findCustomerByEmail(memberEmail);
            if (groupMember != null) {
                groupMemberObjects.add(groupMember);
            }
        }

        // Get devices user has permission to access
        List<Gadget> accessibleGroupDevices =
currentUser.getAccessibleGroupDevices(groupMemberObjects);
        allGadgets.addAll(accessibleGroupDevices);
        groupDeviceCount = accessibleGroupDevices.size();

        // Display group information
        System.out.println("\n=== Group Gadgets ===");
        System.out.println("[INFO] Group size: " + currentUser.getGroupSize() +
                           " member(s) | Admin: " + currentUser.getGroupCreator());
        System.out.println("[INFO] Your role: " +
                           (currentUser.isGroupAdmin() ? "Admin" : "Member"));
        System.out.println("[INFO] Showing your devices + devices you have permission
to access");

        if (groupDeviceCount > 0) {
            System.out.println("[INFO] You have access to " + groupDeviceCount + "
group member devices");
        } else {
            System.out.println("[INFO] No group devices shared with you. Ask admin for
device access permissions.");
        }
    } else {                                // INDIVIDUAL MODE: Show only personal
devices
        allGadgets = currentUser.getGadgets();
        System.out.println("\n=== Your Gadgets ===");
        System.out.println("[INFO] Showing only your personal devices (not part of any
group)");
        System.out.println("[INFO] Use 'Group Management' to create a group and share
devices with others");
    }

    if (allGadgets == null || allGadgets.isEmpty()) {
        System.out.println("No gadgets found! Please connect to some gadgets first.");
        return allGadgets;
    }

    // Ensure all devices have power ratings
    for (Gadget gadget : allGadgets) {
        gadget.ensurePowerRating();
    }

    displayAutoAlignedTable(allGadgets);    // Display formatted table

    return allGadgets;
}


 Intelligent Device Status Control with Group Support (Lines 208-300)
java
public boolean changeGadgetStatus(String gadgetType) {
    if (!sessionManager.isLoggedIn()) {     // Session validation
        System.out.println("Please login first!");
        return false;
    }
```

```
    try {
        if (gadgetType == null || gadgetType.trim().isEmpty()) {
            System.out.println("Gadget type cannot be empty!");
            return false;
        }

        gadgetType = gadgetType.trim().toUpperCase(); // Normalize input

        Customer currentUser = sessionManager.getCurrentUser();
        List<Gadget> gadgets = viewGadgets();   // Get all accessible devices

        if (gadgets == null || gadgets.isEmpty()) {
            System.out.println("No gadgets found! Please connect to some gadgets
first.");
            return false;
        }

        Gadget targetGadget = null;
        Customer gadgetOwner = null;

        // STEP 1: Search in personal devices first
        if (currentUser.getGadgets() != null) {
            for (Gadget gadget : currentUser.getGadgets()) {
                if (gadget.getType().equalsIgnoreCase(gadgetType)) {
                    targetGadget = gadget;
                    gadgetOwner = currentUser;      // User owns this device
                    break;
                }
            }
        }

        // STEP 2: Search in group member devices if not found in personal devices
        if (targetGadget == null && currentUser.isPartOfGroup()) {
            for (String memberEmail : currentUser.getGroupMembers()) {
                Customer member = customerService.findCustomerByEmail(memberEmail);
                if (member != null && member.getGadgets() != null) {
                    for (Gadget gadget : member.getGadgets()) {
                        if (gadget.getType().equalsIgnoreCase(gadgetType)) {
                            targetGadget = gadget;
                            gadgetOwner = member;   // Group member owns this device
                            break;
                        }
                    }
                    if (targetGadget != null) break;
                }
            }
        }

        if (targetGadget == null) {             // Device not found
            System.out.println("Gadget type '" + gadgetType + "' not found!");
            System.out.println("Available gadgets: " + gadgets.stream()
                    .map(Gadget::getType)
                    .distinct()
                    .reduce((a, b) -> a + ", " + b)
                    .orElse("None"));
            return false;
        }

        // STEP 3: Execute status change with energy tracking
        targetGadget.ensurePowerRating();       // Ensure power rating
        String previousStatus = targetGadget.getStatus();
        targetGadget.toggleStatus();            // Toggle with energy calculation
        String newStatus = targetGadget.getStatus();
```

```
        // STEP 4: Persist changes to database
        boolean updated = customerService.updateCustomer(gadgetOwner);

        // STEP 5: Update session if user owns the device
        if (gadgetOwner.getEmail().equals(currentUser.getEmail())) {
            sessionManager.updateCurrentUser(gadgetOwner);
        }

        if (updated) {                          // Success feedback
            if ("ON".equals(newStatus)) {
                System.out.println("[SUCCESS] Switched on successful");
            } else {
                System.out.println("[SUCCESS] Switched off successful");
            }

            System.out.println("\n=== All Gadgets Status ===");
            viewGadgets();                      // Show updated status
            return true;
        } else {                                // Rollback on failure
            targetGadget.setStatus(previousStatus);
            System.out.println("[ERROR] Failed to update gadget status!");
            return false;
        }

    } catch (Exception e) {
        System.out.println("Error changing gadget status. Please try again.");
        return false;
    }
}


 Advanced Auto-Aligned Table Display System (Lines 1119-1281)
java
private void displayAutoAlignedTable(List<Gadget> allGadgets) {
    TableDimensions dimensions = calculateTableDimensions(allGadgets);
    TableFormatStrings formats = createTableFormatStrings(dimensions);

    System.out.println("Device List (Enter number to view detailed energy info):");
    System.out.println(formats.borderFormat);
    System.out.printf(formats.headerFormat + "\n", "", "Device", "Power", "Status",
"Usage Time", "Energy(kWh)");
    System.out.println(formats.borderFormat);

    displayTableRows(allGadgets, formats);
    System.out.println(formats.borderFormat);
}

private TableDimensions calculateTableDimensions(List<Gadget> allGadgets) {
    // DYNAMIC COLUMN WIDTH CALCULATION
    int numWidth = Math.max(2, String.valueOf(allGadgets.size()).length());
    int deviceWidth = "Device".length();
    int powerWidth = "Power".length();
    int statusWidth = "Status".length();
    int usageWidth = "Usage Time".length();
    int energyWidth = "Energy(kWh)".length();

    LocalDateTime now = LocalDateTime.now();
    final DateTimeFormatter timeFormatter = DateTimeFormatter.ofPattern("dd-MM
HH:mm");

    // CALCULATE MAXIMUM REQUIRED WIDTH FOR EACH COLUMN
    for (Gadget gadget : allGadgets) {
```

```
        String deviceName = String.format("%s %s (%s)", gadget.getType(),
gadget.getModel(), gadget.getRoomName());
        deviceWidth = Math.max(deviceWidth, deviceName.length());

        String powerStr = String.format("%.0fW", gadget.getPowerRatingWatts());
        powerWidth = Math.max(powerWidth, powerStr.length());

        String statusDisplay = gadget.isOn() ? "RUNNING" : "OFF";
        statusWidth = Math.max(statusWidth, statusDisplay.length());

        String usageTime = gadget.getCurrentUsageTimeFormatted();
        usageWidth = Math.max(usageWidth, usageTime.length());

        String energyStr = String.format("%.3f",
gadget.getCurrentTotalEnergyConsumedKWh());
        energyWidth = Math.max(energyWidth, energyStr.length());

        // ACCOUNT FOR ADDITIONAL DISPLAY ELEMENTS
        if (gadget.isOn() && gadget.getLastOnTime() != null) {
            deviceWidth = Math.max(deviceWidth, "  Current Session:".length());
            String sessionTime = String.format("%.1fh",
gadget.getCurrentSessionUsageHours());
            usageWidth = Math.max(usageWidth, sessionTime.length());
        }

        // ACCOUNT FOR TIMER INFORMATION DISPLAY
        if (gadget.isTimerEnabled()) {
            String timerInfo = buildTimerInfo(gadget, now, timeFormatter);
            String timerDisplay = "  Timer: " + timerInfo;
            deviceWidth = Math.max(deviceWidth, timerDisplay.length());
        }
    }

    return new TableDimensions(
        numWidth,
        Math.max(deviceWidth, 25),  // Minimum device width for readability
        Math.max(powerWidth, 8),    // Minimum power width
        Math.max(statusWidth, 9),   // Minimum status width
        Math.max(usageWidth, 11),   // Minimum usage width
        Math.max(energyWidth, 11)   // Minimum energy width
    );
}

// INNER CLASSES FOR TABLE FORMATTING
private static class TableDimensions {
    final int numWidth, deviceWidth, powerWidth, statusWidth, usageWidth, energyWidth;

    TableDimensions(int numWidth, int deviceWidth, int powerWidth, int statusWidth,
int usageWidth, int energyWidth) {
        this.numWidth = numWidth;
        this.deviceWidth = deviceWidth;
        this.powerWidth = powerWidth;
        this.statusWidth = statusWidth;
        this.usageWidth = usageWidth;
        this.energyWidth = energyWidth;
    }
}

private static class TableFormatStrings {
    final String borderFormat, headerFormat, rowFormat, emptyRowFormat;

    TableFormatStrings(String borderFormat, String headerFormat, String rowFormat,
String emptyRowFormat) {
```

```
        this.borderFormat = borderFormat;
        this.headerFormat = headerFormat;
        this.rowFormat = rowFormat;
        this.emptyRowFormat = emptyRowFormat;
    }
}
```

EnergyManagementService.java - Advanced Energy Monitoring {energy-service}

```
Package Declaration and Core Imports (Lines 1-10)
java
package com.smarthome.service;                // Service layer package

import com.smarthome.model.Customer;          // User entity for energy calculations
import com.smarthome.model.Gadget;            // Device entity for energy tracking
import com.smarthome.model.DeletedDeviceEnergyRecord; // Preserved energy data

import java.time.LocalDateTime;               // Date/time operations for reporting
import java.time.format.DateTimeFormatter;   // Time formatting for displays
import java.util.List;                        // List interface for device collections


Energy Report Data Structure (Lines 11-30)
java
public class EnergyManagementService {        // Advanced energy calculation service

    public static class EnergyReport {        // Energy report container class
        private double totalEnergyKWh;        // Total energy consumption in kWh
        private double totalCostRupees;       // Total cost in Indian Rupees
        private String reportPeriod;          // Report time period description
        private List<Gadget> devices;         // List of devices included in report

        public EnergyReport(double totalEnergyKWh, double totalCostRupees,
                            String reportPeriod, List<Gadget> devices) {
            this.totalEnergyKWh = totalEnergyKWh;     // Set total energy
            this.totalCostRupees = totalCostRupees;   // Set total cost
            this.reportPeriod = reportPeriod;         // Set period description
            this.devices = devices;                   // Set device list
        }

        // Getters for accessing report data
        public double getTotalEnergyKWh() { return totalEnergyKWh; }
        public double getTotalCostRupees() { return totalCostRupees; }
        public String getReportPeriod() { return reportPeriod; }
        public List<Gadget> getDevices() { return devices; }
    }


 Comprehensive Energy Report Generation (Lines 32-53)
java
public EnergyReport generateEnergyReport(Customer customer) {
    List<Gadget> devices = customer.getGadgets();    // Get user's devices
    double totalEnergyKWh = 0.0;                     // Initialize total energy

    for (Gadget device : devices) {                  // Process each device
        double currentSessionEnergy = 0.0;           // Current session energy
        if (device.isOn() && device.getLastOnTime() != null) {
            // Calculate energy for devices currently running
            double currentSessionHours = device.getCurrentSessionUsageHours();
            currentSessionEnergy = (device.getPowerRatingWatts() / 1000.0)
currentSessionHours;
            // Line 40: Power(kW) × Time(hours) = Energy(kWh)
        }
```

```
        totalEnergyKWh += device.getTotalEnergyConsumedKWh() + currentSessionEnergy;
        // Line 43: Add historical energy + current session energy
    }

    // CRITICAL: Include energy from deleted devices for accurate billing
    double deletedDeviceEnergy =
customer.getTotalDeletedDeviceEnergyForCurrentMonth();
    totalEnergyKWh += deletedDeviceEnergy;
    // Line 47: Ensures billing accuracy even after device deletions

    double totalCost = calculateSlabBasedCost(totalEnergyKWh);
    String reportPeriod = "Monthly Report - " +
        LocalDateTime.now().format(DateTimeFormatter.ofPattern("MMMM yyyy"));

    return new EnergyReport(totalEnergyKWh, totalCost, reportPeriod, devices);
}


 Indian Electricity Tariff Slab-Based Cost Calculation (Lines 55-73)
java
public double calculateSlabBasedCost(double totalKWh) {
    double totalCost = 0.0;                          // Initialize total cost

    // PROGRESSIVE SLAB STRUCTURE (Based on Indian Electricity Board rates)
    if (totalKWh <= 30) {                            // SLAB 1: 0-30 kWh
        totalCost = totalKWh  1.90;                 // Rs. 1.90 per kWh (subsidized
rate)
    } else if (totalKWh <= 75) {                     // SLAB 2: 31-75 kWh
        totalCost = 30  1.90 + (totalKWh - 30)  3.0;
        // Line 61: First 30 at Rs. 1.90, remainder at Rs. 3.0
    } else if (totalKWh <= 125) {                    // SLAB 3: 76-125 kWh
        totalCost = 30  1.90 + 45  3.0 + (totalKWh - 75)  4.50;
        // Line 63: Cumulative slab calculation with Rs. 4.50 rate
    } else if (totalKWh <= 225) {                    // SLAB 4: 126-225 kWh
        totalCost = 30  1.90 + 45  3.0 + 50  4.50 + (totalKWh - 125)  6.0;
        // Line 65: Higher consumption at Rs. 6.0 per kWh
    } else if (totalKWh <= 400) {                    // SLAB 5: 226-400 kWh
        totalCost = 30  1.90 + 45  3.0 + 50  4.50 + 100  6.0 + (totalKWh - 225)  8.75;
        // Line 67: Heavy usage at Rs. 8.75 per kWh
    } else {                                         // SLAB 6: 400+ kWh
        totalCost = 30  1.90 + 45  3.0 + 50  4.50 + 100  6.0 + 175  8.75 + (totalKWh -
400)  9.75;
        // Line 69: Maximum rate Rs. 9.75 per kWh for excessive consumption
    }

    return Math.round(totalCost  100.0) / 100.0;  // Round to 2 decimal places
}


 Detailed Slab Breakdown Table Generation (Lines 75-136)
java
public String getSlabBreakdown(double totalKWh) {
    StringBuilder breakdown = new StringBuilder();
    breakdown.append("\n=== Energy Cost Breakdown (Slab-wise) ===\n");
    breakdown.append("+---------------+----------+----------+----------+\n");
    breakdown.append(String.format("| %-13s | %-8s | %-8s | %-8s |\n",
                     "Slab Range", "Units", "Rate/Unit", "Cost"));
    breakdown.append("+---------------+----------+----------+----------+\n");

    double remainingKWh = totalKWh;                  // Track remaining consumption
    double totalCost = 0.0;                         // Track cumulative cost
```

```
    // SLAB 1: 0-30 kWh at Rs. 1.90
    if (remainingKWh > 0) {
        double slab1 = Math.min(remainingKWh, 30);  // Up to 30 kWh
        double cost1 = slab1  1.90;              // Calculate cost for this slab
        totalCost += cost1;                       // Add to total
        breakdown.append(String.format("| %-13s | %8.2f | %8.2f | %8.2f |\n",
                        "0-30 kWh", slab1, 1.90, cost1));
        remainingKWh -= slab1;                    // Reduce remaining consumption
    }

    // SLAB 2: 31-75 kWh at Rs. 3.0
    if (remainingKWh > 0) {
        double slab2 = Math.min(remainingKWh, 45); // Next 45 kWh (31-75)
        double cost2 = slab2  3.0;
        totalCost += cost2;
        breakdown.append(String.format("| %-13s | %8.2f | %8.2f | %8.2f |\n",
                        "31-75 kWh", slab2, 3.0, cost2));
        remainingKWh -= slab2;
    }

    // SLAB 3: 76-125 kWh at Rs. 4.50
    if (remainingKWh > 0) {
        double slab3 = Math.min(remainingKWh, 50); // Next 50 kWh (76-125)
        double cost3 = slab3  4.50;
        totalCost += cost3;
        breakdown.append(String.format("| %-13s | %8.2f | %8.2f | %8.2f |\n",
                        "76-125 kWh", slab3, 4.50, cost3));
        remainingKWh -= slab3;
    }

    // SLAB 4: 126-225 kWh at Rs. 6.0
    if (remainingKWh > 0) {
        double slab4 = Math.min(remainingKWh, 100); // Next 100 kWh (126-225)
        double cost4 = slab4  6.0;
        totalCost += cost4;
        breakdown.append(String.format("| %-13s | %8.2f | %8.2f | %8.2f |\n",
                        "126-225 kWh", slab4, 6.0, cost4));
        remainingKWh -= slab4;
    }

    // SLAB 5: 226-400 kWh at Rs. 8.75
    if (remainingKWh > 0) {
        double slab5 = Math.min(remainingKWh, 175); // Next 175 kWh (226-400)
        double cost5 = slab5  8.75;
        totalCost += cost5;
        breakdown.append(String.format("| %-13s | %8.2f | %8.2f | %8.2f |\n",
                        "226-400 kWh", slab5, 8.75, cost5));
        remainingKWh -= slab5;
    }

    // SLAB 6: 400+ kWh at Rs. 9.75
    if (remainingKWh > 0) {
        double cost6 = remainingKWh  9.75;        // All remaining at highest rate
        totalCost += cost6;
        breakdown.append(String.format("| %-13s | %8.2f | %8.2f | %8.2f |\n",
                        "400+ kWh", remainingKWh, 9.75, cost6));
    }

    // TOTAL ROW
    breakdown.append("+---------------+----------+----------+----------+\n");
    breakdown.append(String.format("| %-13s | %8.2f | %-8s | %8.2f |\n",
                    "TOTAL", totalKWh, "-", totalCost));
    breakdown.append("+---------------+----------+----------+----------+\n");
```

```
    return breakdown.toString();                    // Return formatted breakdown
}


 Comprehensive Device Energy Usage Display (Lines 138-195)
java
public void displayDeviceEnergyUsage(Customer customer) {
    List<Gadget> devices = customer.getGadgets();

    System.out.println("\n=== Device Energy Usage Details ===");
    System.out.println("+-------------------------+---------+---------+-------------+-
------------+-------------+");
    System.out.printf("| %-23s | %-7s | %-7s | %-11s | %-11s | %-11s |\n",
                      "Device", "Power", "Status", "Usage Time", "Energy(kWh)",
"Cost(Rs.)");
    System.out.println("+-------------------------+---------+---------+-------------+-
------------+-------------+");

    // Display active and historical devices
    for (Gadget device : devices) {
        double totalEnergy = device.getTotalEnergyConsumedKWh(); // Historical energy

        if (device.isOn() && device.getLastOnTime() != null) {
            // Add current session energy for running devices
            double currentSessionHours = device.getCurrentSessionUsageHours();
            double currentSessionEnergy = (device.getPowerRatingWatts() / 1000.0)
currentSessionHours;
            totalEnergy += currentSessionEnergy;    // Real-time total energy
        }

        double deviceCost = calculateSlabBasedCost(totalEnergy); // Individual device
cost
        String status = device.isOn() ? "RUNNING" : "OFF";
        String deviceName = String.format("%s %s (%s)", device.getType(),
device.getModel(), device.getRoomName());
        if (deviceName.length() > 23) {            // Truncate long names
            deviceName = deviceName.substring(0, 20) + "...";
        }

        System.out.printf("| %-23s | %7.0fW | %-7s | %11s | %11.3f | %11.2f |\n",
                          deviceName,
                          device.getPowerRatingWatts(),
                          status,
                          device.getCurrentUsageTimeFormatted(),
                          totalEnergy,
                          deviceCost);

        // Show current session details for running devices
        if (device.isOn() && device.getLastOnTime() != null) {
            System.out.printf("| %-23s | %-7s | %-7s | %-11s | %-11s | %-11s |\n",
                              "  Current Session:", "", "",
                              String.format("%.2fh",
device.getCurrentSessionUsageHours()), "", "");
        }
    }

    // CRITICAL: Display deleted devices energy consumption
    double deletedDeviceEnergy =
customer.getTotalDeletedDeviceEnergyForCurrentMonth();
    if (deletedDeviceEnergy > 0) {
        System.out.println("+-------------------------+---------+---------+-----------
--+-------------+-------------+");
```

```
        double deletedDeviceCost = calculateSlabBasedCost(deletedDeviceEnergy);
        System.out.printf("| %-23s | %-7s | %-7s | %-11s | %11.3f | %11.2f |\n",
                          "[DELETED DEVICES]", "-", "DELETED", "-",
deletedDeviceEnergy, deletedDeviceCost);
        System.out.printf("| %-23s | %-7s | %-7s | %-11s | %-11s | %-11s |\n",
                          "  (Historical data)", "", "", "", "", "");
    }

    System.out.println("+-------------------------+---------+---------+-------------+-
------------+-------------+");

    // Show detailed breakdown of deleted devices if any exist
    if (deletedDeviceEnergy > 0) {
        displayDeletedDeviceBreakdown(customer);   // Show detailed deleted device
info
    }
}


 Deleted Device Energy Breakdown Display (Lines 197-230)
java
public void displayDeletedDeviceBreakdown(Customer customer) {
    LocalDateTime now = LocalDateTime.now();
    String currentMonth = now.getYear() + "-" + String.format("%02d",
now.getMonthValue());
    // Line 199: Format as "YYYY-MM" for month filtering

    List<DeletedDeviceEnergyRecord> deletedDevices =
customer.getDeletedDeviceEnergyRecords();
    List<DeletedDeviceEnergyRecord> currentMonthDeleted = deletedDevices.stream()
            .filter(record -> currentMonth.equals(record.getDeletionMonth()))
            .toList();
    // Lines 202-204: Filter records for current month only

    if (!currentMonthDeleted.isEmpty()) {
        System.out.println("\n=== Deleted Devices Energy Breakdown (This Month) ===");
        System.out.println("+-------------------------+---------+-------------+-------
------+----------------------+");
        System.out.printf("| %-23s | %-7s | %-11s | %-11s | %-20s |\n",
                          "Deleted Device", "Power", "Usage Time", "Energy(kWh)",
"Deletion Date");
        System.out.println("+-------------------------+---------+-------------+-------
------+----------------------+");

        for (DeletedDeviceEnergyRecord record : currentMonthDeleted) {
            String deviceName = String.format("%s %s (%s)",
                record.getDeviceType(), record.getDeviceModel(),
record.getRoomName());
            if (deviceName.length() > 23) {        // Truncate long device names
                deviceName = deviceName.substring(0, 20) + "...";
            }

            String deletionDate =
record.getDeletionTime().format(DateTimeFormatter.ofPattern("dd-MM HH:mm"));

            System.out.printf("| %-23s | %7.0fW | %-11s | %11.3f | %-20s |\n",
                              deviceName,
                              record.getPowerRatingWatts(),
                              record.getFormattedUsageTime(),
                              record.getTotalEnergyConsumedKWh(),
                              deletionDate);
        }
```

```
        System.out.println("+-------------------------+---------+-------------+-------
------+----------------------+");
    }
}


 Energy Efficiency Tips and Recommendations (Lines 232-254)
java
public String getEnergyEfficiencyTips(double totalKWh) {
    StringBuilder tips = new StringBuilder();
    tips.append("\n=== Energy Efficiency Tips ===\n");

    // USAGE-BASED RECOMMENDATIONS
    if (totalKWh > 300) {                          // HIGH USAGE ALERT
        tips.append("[HIGH USAGE ALERT] Your consumption is quite high!\n");
        tips.append("- Consider using Timer functions to auto-schedule device
operations\n");
        tips.append("- Turn off devices when not in use\n");
        tips.append("- Use energy-efficient appliances\n");
    } else if (totalKWh > 150) {                   // MODERATE USAGE
        tips.append("[MODERATE USAGE] You're doing well, but there's room for
improvement!\n");
        tips.append("- Use Timer scheduling for AC and Geyser during off-peak
hours\n");
        tips.append("- Consider LED lighting for better efficiency\n");
    } else {                                       // EXCELLENT USAGE
        tips.append("[EXCELLENT] Great job on managing your energy consumption!\n");
        tips.append("- Keep up the good work with energy-conscious usage\n");
    }

    // GENERAL ENERGY SAVING TIPS
    tips.append("- Peak hours (6-10 PM): Avoid using high-power devices\n");
    tips.append("- Use our Calendar Events to schedule energy-intensive
operations\n");

    return tips.toString();                       // Return personalized
recommendations
}
```

TimerService.java - Background Automation Service {timer-service}

```
Package Declaration and Concurrency Imports (Lines 1-16)
java
package com.smarthome.service;                 // Service layer package

import com.smarthome.model.Customer;           // User entity for timer operations
import com.smarthome.model.Gadget;             // Device entity for automation
import com.smarthome.util.SessionManager;      // Session management for current user

import java.time.LocalDateTime;                // Date/time operations for scheduling
import java.time.format.DateTimeFormatter;    // Time formatting for displays
import java.time.format.DateTimeParseException; // Date parsing exception handling
import java.time.temporal.ChronoUnit;          // Time calculations for countdowns
import java.util.ArrayList;                    // Dynamic collections
import java.util.List;                         // List interface
import java.util.concurrent.Executors;        // Thread pool creation
import java.util.concurrent.ScheduledExecutorService; // Background task scheduling
import java.util.concurrent.TimeUnit;         // Time unit constants


 Singleton Timer Service Class Structure (Lines 17-34)
java
```

```
public class TimerService {                    // Background automation service

    private final ScheduledExecutorService scheduler; // Background task scheduler
    private final CustomerService customerService;    // Customer data access
    private static TimerService instance;             // Singleton instance

    private TimerService(CustomerService customerService) {
        this.scheduler = Executors.newScheduledThreadPool(5);
        // Line 24: Creates thread pool with 5 concurrent timer execution threads
        this.customerService = customerService;       // Store customer service
reference
        startTimerMonitoring();                       // Start background monitoring
        // Line 26: Immediately begin checking for due timers
    }

    public static synchronized TimerService getInstance(CustomerService
customerService) {
        if (instance == null) {                       // Lazy initialization
            instance = new TimerService(customerService);
        }
        return instance;                              // Return singleton instance
        // Lines 29-33: Thread-safe singleton pattern with synchronization
    }


 Timer Task Data Structure (Lines 36-56)
java
public static class TimerTask {               // Timer task container class
    private final String deviceType;          // Device type to control
    private final String roomName;            // Room location of device
    private final String action;              // Action to perform (ON/OFF)
    private final LocalDateTime scheduledTime; // When to execute
    private final String userEmail;           // Owner of the timer

    public TimerTask(String deviceType, String roomName, String action,
                     LocalDateTime scheduledTime, String userEmail) {
        this.deviceType = deviceType;          // Set device type
        this.roomName = roomName;              // Set room location
        this.action = action;                  // Set action (ON/OFF)
        this.scheduledTime = scheduledTime;    // Set execution time
        this.userEmail = userEmail;            // Set timer owner
    }

    // Getters for accessing task data
    public String getDeviceType() { return deviceType; }
    public String getRoomName() { return roomName; }
    public String getAction() { return action; }
    public LocalDateTime getScheduledTime() { return scheduledTime; }
    public String getUserEmail() { return userEmail; }
}


 Smart Timer Scheduling with Comprehensive Validation (Lines 58-107)
java
public boolean scheduleDeviceTimer(Customer customer, String deviceType, String
roomName,
                                   String action, LocalDateTime scheduledTime) {
    try {
        Gadget device = customer.findGadget(deviceType, roomName);
        if (device == null) {                         // Device existence check
            System.out.println("[ERROR] Device not found: " + deviceType + " in " +
roomName);
            return false;
```

```
        }

        LocalDateTime now = LocalDateTime.now();
        if (scheduledTime.isBefore(now)) {          // Past time validation
            System.out.printf("[ERROR] Cannot schedule timer for past time!\n");
            System.out.printf("Current time: %s\n",
                now.format(DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm")));
            System.out.printf("Requested time: %s\n",
                scheduledTime.format(DateTimeFormatter.ofPattern("dd-MM-yyyy
HH:mm")));
            return false;
        }

        if (scheduledTime.isBefore(now.plusMinutes(1))) { // Minimum future time check
            System.out.printf("[ERROR] Timer must be scheduled at least 1 minute in
the future!\n");
            System.out.printf("Minimum allowed time: %s\n",
                now.plusMinutes(1).format(DateTimeFormatter.ofPattern("dd-MM-yyyy
HH:mm")));
            return false;
        }

        // SET TIMER BASED ON ACTION TYPE
        if (action.equalsIgnoreCase("ON")) {
            device.setScheduledOnTime(scheduledTime);   // Set ON timer
        } else if (action.equalsIgnoreCase("OFF")) {
            device.setScheduledOffTime(scheduledTime);  // Set OFF timer
        } else {
            System.out.println("[ERROR] Invalid action! Use 'ON' or 'OFF'");
            return false;
        }

        device.setTimerEnabled(true);                   // Enable timer flag

        boolean updated = customerService.updateCustomer(customer);
        if (updated) {                                  // Database update successful
            System.out.println("[SUCCESS] Timer scheduled for " + device.getType() + "
" +
                               device.getModel() + " in " + device.getRoomName() + " to
turn " +
                               action.toUpperCase() + " at " +
                               scheduledTime.format(DateTimeFormatter.ofPattern("dd-MM-
yyyy HH:mm")));
            return true;
        } else {
            System.out.println("[ERROR] Failed to save timer schedule!");
            return false;
        }

    } catch (Exception e) {
        System.out.println("[ERROR] Error scheduling timer: " + e.getMessage());
        return false;
    }
}


 Advanced Timer Display with Real-Time Countdown (Lines 109-163)
java
public void displayScheduledTimers(Customer customer) {
    forceTimerCheck();                              // Check for expired timers first

    System.out.println("\n=== Scheduled Timers ===");
```

```
    List<Gadget> devicesWithTimers = new ArrayList<>();
    for (Gadget device : customer.getGadgets()) {
        if (device.isTimerEnabled() &&
            (device.getScheduledOnTime() != null || device.getScheduledOffTime() !=
null)) {
            devicesWithTimers.add(device);          // Collect devices with active
timers
        }
    }

    if (devicesWithTimers.isEmpty()) {
        System.out.println("No timers scheduled.");
        return;
    }

    LocalDateTime now = LocalDateTime.now();

    // FORMATTED TABLE DISPLAY
    System.out.println("+----+-------------------------+--------+------------------+-
---------------------+");
    System.out.printf("| %-2s | %-23s | %-6s | %-17s | %-20s |\n",
                      "", "Device", "Action", "Scheduled Time", "Status");
    System.out.println("+----+-------------------------+--------+------------------+-
---------------------+");

    int timerIndex = 1;
    for (Gadget device : devicesWithTimers) {
        String deviceName = String.format("%s %s (%s)", device.getType(),
device.getModel(), device.getRoomName());
        if (deviceName.length() > 23) {            // Truncate long device names
            deviceName = deviceName.substring(0, 20) + "...";
        }

        // DISPLAY ON TIMER
        if (device.getScheduledOnTime() != null) {
            String countdown = getCountdownString(now, device.getScheduledOnTime());
            String scheduledTime =
device.getScheduledOnTime().format(DateTimeFormatter.ofPattern("dd-MM HH:mm"));
            String status = countdown.length() > 20 ? countdown.substring(0, 17) +
"..." : countdown;

            System.out.printf("| %-2d | %-23s | %-6s | %-17s | %-20s |\n",
                              timerIndex++, deviceName, "ON", scheduledTime, status);
        }

        // DISPLAY OFF TIMER
        if (device.getScheduledOffTime() != null) {
            String countdown = getCountdownString(now, device.getScheduledOffTime());
            String scheduledTime =
device.getScheduledOffTime().format(DateTimeFormatter.ofPattern("dd-MM HH:mm"));
            String status = countdown.length() > 20 ? countdown.substring(0, 17) +
"..." : countdown;

            String displayDeviceName = device.getScheduledOnTime() != null ? "" :
deviceName;
            // Line 155: Don't repeat device name if ON timer already shown

            System.out.printf("| %-2s | %-23s | %-6s | %-17s | %-20s |\n",
                              device.getScheduledOnTime() != null ? "" :
String.valueOf(timerIndex++),
                              displayDeviceName, "OFF", scheduledTime, status);
        }
    }
```

```
    System.out.println("+----+-------------------------+--------+-------------------+-
----------------------+");
}

Intelligent Countdown String Generation (Lines 165-193)
java
private String getCountdownString(LocalDateTime now, LocalDateTime scheduledTime) {
    if (scheduledTime.isBefore(now)) {              // Timer is overdue
        long minutesOverdue = ChronoUnit.MINUTES.between(scheduledTime, now);
        if (minutesOverdue <= 10) {
            return "[EXECUTING/DUE]";               // Within execution window
        } else {
            return "[EXPIRED]";                     // Expired and removed
        }
    }

    // CALCULATE REMAINING TIME
    long totalSeconds = ChronoUnit.SECONDS.between(now, scheduledTime);
    long totalMinutes = totalSeconds / 60;
    long seconds = totalSeconds % 60;
    long days = totalMinutes / (24  60);
    long hours = (totalMinutes % (24  60)) / 60;
    long minutes = totalMinutes % 60;

    // SMART TIME FORMATTING
    if (totalMinutes == 0 && seconds <= 60) {
        return String.format("[%ds remaining]", seconds);    // Show seconds for
imminent timers
    } else if (totalMinutes < 2) {
        return String.format("[%dm %ds remaining]", minutes, seconds); // Minutes and
seconds
    } else if (days > 0) {
        return String.format("[%dd %dh %dm remaining]", days, hours, minutes); // Full
format
    } else if (hours > 0) {
        return String.format("[%dh %dm remaining]", hours, minutes); // Hours and
minutes
    } else {
        return String.format("[%dm remaining]", minutes);   // Minutes only
    }
}


 CRITICAL: Background Timer Monitoring System (Lines 232-344)
java
private void startTimerMonitoring() {               // Start background monitoring
    scheduler.scheduleAtFixedRate(() -> {           // Execute every 10 seconds
        try {
            checkAndExecuteScheduledTasks();        // Check for due timers
        } catch (Exception e) {
            System.err.println("Error in timer monitoring: " + e.getMessage());
        }
    }, 0, 10, TimeUnit.SECONDS);                   // 0 initial delay, 10-second
intervals
}

private void checkAndExecuteScheduledTasks() {
    LocalDateTime now = LocalDateTime.now();

    try {
        List<Customer> allCustomers = getAllCustomersWithTimers(); // Get customers
with timers
```

```
        for (Customer customer : allCustomers) {
            boolean customerUpdated = false;        // Track if customer needs
database update

            for (Gadget device : customer.getGadgets()) {
                if (!device.isTimerEnabled()) continue; // Skip devices without timers

                // CHECK ON TIMERS
                if (device.getScheduledOnTime() != null) {
                    LocalDateTime scheduledOnTime = device.getScheduledOnTime();

                    if (now.isAfter(scheduledOnTime) || now.isEqual(scheduledOnTime))
{
                        long minutesSinceScheduled =
ChronoUnit.MINUTES.between(scheduledOnTime, now);

                        if (minutesSinceScheduled <= 10) {  // Within execution window
                            String previousStatus = device.getStatus();
                            device.turnOn();                // Execute ON command
                            String newStatus = device.getStatus();

                            device.setScheduledOnTime(null); // Clear executed timer
                            customerUpdated = true;         // Mark for database
update

                            if (device.getScheduledOffTime() == null) {
                                device.setTimerEnabled(false); // Disable if no other
timers
                            }

                            // USER NOTIFICATION
                            System.out.println("\n[TIMER EXECUTED] " +
device.getType() + " " +
                                               device.getModel() + " in " +
device.getRoomName() +
                                               " turned ON automatically");
                            System.out.println("  Scheduled: " +

scheduledOnTime.format(DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm")));
                            System.out.println("  Executed: " +
                                now.format(DateTimeFormatter.ofPattern("dd-MM-yyyy
HH:mm:ss")));
                            System.out.println("  Status: " + previousStatus + " → " +
newStatus);
                            System.out.print("\nPress Enter to continue or enter your
choice: ");

                        } else {                            // Expired timer cleanup
                            device.setScheduledOnTime(null);
                            customerUpdated = true;

                            if (device.getScheduledOffTime() == null) {
                                device.setTimerEnabled(false);
                            }

                            System.out.println("[TIMER EXPIRED] Old ON timer removed
for " +
                                               device.getType() + " in " +
device.getRoomName());
                        }
                    }
                }
```

```
                // CHECK OFF TIMERS (Similar logic for OFF commands)
                if (device.getScheduledOffTime() != null) {
                    LocalDateTime scheduledOffTime = device.getScheduledOffTime();

                    if (now.isAfter(scheduledOffTime) ||
now.isEqual(scheduledOffTime)) {
                        long minutesSinceScheduled =
ChronoUnit.MINUTES.between(scheduledOffTime, now);

                        if (minutesSinceScheduled <= 10) {  // Within execution window
                            String previousStatus = device.getStatus();
                            device.turnOff();               // Execute OFF command
                            String newStatus = device.getStatus();

                            device.setScheduledOffTime(null); // Clear executed timer
                            customerUpdated = true;

                            if (device.getScheduledOnTime() == null) {
                                device.setTimerEnabled(false);
                            }

                            System.out.println("\n[TIMER EXECUTED] " +
device.getType() + " " +
                                               device.getModel() + " in " +
device.getRoomName() +
                                               " turned OFF automatically");
                            System.out.println("  Scheduled: " +

scheduledOffTime.format(DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm")));
                            System.out.println("  Executed: " +
                                now.format(DateTimeFormatter.ofPattern("dd-MM-yyyy
HH:mm:ss")));
                            System.out.println("  Status: " + previousStatus + " → " +
newStatus);
                            System.out.print("\nPress Enter to continue or enter your
choice: ");

                        } else {                            // Expired timer cleanup
                            device.setScheduledOffTime(null);
                            customerUpdated = true;

                            if (device.getScheduledOnTime() == null) {
                                device.setTimerEnabled(false);
                            }

                            System.out.println("[TIMER EXPIRED] Old OFF timer removed
for " +
                                               device.getType() + " in " +
device.getRoomName());
                        }
                    }
                }
            }

            // PERSIST CHANGES TO DATABASE
            if (customerUpdated) {
                boolean saveSuccess = customerService.updateCustomer(customer);
                if (!saveSuccess) {
                    System.err.println("[ERROR] Failed to save device state changes
after timer execution");
                }
            }
        }
```

```
    } catch (Exception e) {
        System.err.println("Error checking scheduled tasks: " + e.getMessage());
        e.printStackTrace();
    }
}


 Service Lifecycle Management (Lines 387-400)
java
public void shutdown() {                            // Graceful service shutdown
    if (scheduler != null && !scheduler.isShutdown()) {
        scheduler.shutdown();                       // Stop accepting new tasks
        try {
            if (!scheduler.awaitTermination(60, TimeUnit.SECONDS)) {
                scheduler.shutdownNow();            // Force shutdown if needed
                // Line 391: Wait up to 60 seconds for graceful shutdown
            }
        } catch (InterruptedException e) {          // Handle interrupt during
shutdown
            scheduler.shutdownNow();                // Force immediate shutdown
            Thread.currentThread().interrupt();    // Restore interrupt status
        }
    }
}
```

Security Implementation Summary

Authentication Security Layers
1. BCrypt Password Hashing: Industry-standard with salt generation
2. Common Password Blocking: 35+ blocked passwords including regional variants
3. Progressive Lockout Policy: 2→5min, 3→15min, 5→1hr, 7→4hr, 10→24hr
4. Rate Limiting: 1s→3s→5s→10s→15s delays for brute force prevention
5. Security Logging: Detailed attempt tracking with timestamps
6. Account Enumeration Protection: Rate limiting for non-existent accounts

Input Handling Security
1. Consistent Scanner Usage: Single Scanner instance prevents buffer conflicts
2. Navigation Command Support: '0' for menu return throughout application
3. Input Validation: Comprehensive validation for all user inputs
4. Exception Handling: Graceful error handling with user-friendly messages

---

Production Readiness Checklist

Security
- ✅ BCrypt Password Hashing - Industry standard implementation
- ✅ Progressive Account Lockout - Multi-tier security with up to 24-hour lockouts
- ✅ Rate Limiting - Brute force attack prevention
- ✅ Input Validation - Comprehensive validation throughout application
- ✅ Session Management - Thread-safe user state handling

Functionality
✅ 19 Menu Options - Complete feature set across 5 categories
✅ 15+ Device Types - Comprehensive smart home device support
✅ Group Collaboration - Multi-user device sharing with permissions
✅ Energy Monitoring - Real-time consumption with Indian tariff calculation
✅ Background Automation - Timer service with scene management

Performance
✅ Java 21 LTS - Latest long-term support with performance optimizations
✅ Thread Safety - Concurrent operation support
✅ Database Optimization - Efficient DynamoDB operations
✅ Memory Management - Proper resource cleanup and leak prevention

User Experience
✅ Single-Enter Navigation - Consistent input handling throughout
✅ Clear Error Messages - Specific, actionable error feedback
✅ Graceful Shutdown - Proper cleanup on application termination
✅ Demo Mode - Seamless operation without database connectivity

---

Total Code Analysis Summary

Comprehensive Coverage
- 📊 Project Scale: 25+ files across multiple layers (See [Project File Structure](project-file-structure))
- ☕ Java Implementation: 18 files totaling 10,111+ lines of production code
- 🧪 Test Coverage: 2 comprehensive unit test files (300+ lines)
- ⚙️ Configuration: 4 files (Maven POM, properties, dependency management)
- 📖 Documentation: 3 detailed documentation files
- 🏠 Device Support: 400+ device brands across 15+ categories
- 🏢 Location Support: 30+ room types for comprehensive organization
- 🗄️ Database Integration: Full DynamoDB local and AWS cloud support

Architecture Excellence

- Clean MVC Architecture: Clear separation between models, views, and controllers
- Service Layer Pattern: Business logic encapsulation in dedicated services
- Singleton Pattern: Shared services with thread-safe implementations
- Enhanced Security Model: Multi-layer authentication with progressive lockout
- Modern Java 21 Features: Latest LTS optimizations and performance improvements

This comprehensive line-by-line analysis covers every significant line of code across the entire IoT Smart Home Dashboard application, providing complete understanding of implementation details, security measures, business logic, and production-ready features.