Software Requirement Specification (SRS)

For

SMART HOME AUTOMATION SYSTEM

Version 2.0

May 28, 2025

SUBMITTED TO: MAM JAWERIA KANWAL

Prepared By: MUHAMMAD TAIMOOR

NUML-F22-15373

NATIONAL UNIVERSITY OF MODERN LANGUAGES

TABLE OF CONTENT

1. Introduction

1.1 Purpose

The Smart Home Automation System allows users to remotely control home devices such as lights, fans, door locks, and room temperature through a mobile or web interface. It enhances convenience, security, and energy efficiency.

1.2 Scope

This system enables:

- Remote control of household devices.
- Automation via scheduled tasks.
- Security system including **door locking** and **camera monitoring**.
- **Temperature control** for air conditioning and heating.
- Real-time device status updates.
- Secure authentication and user management.

2. Functional Requirements

2.1 User Authentication

- Users must register and log in securely.
- Two-factor authentication for security.

2.2 Device Control

- Users can turn **lights ON/OFF** and adjust brightness.
- Users can turn **fan ON/OFF** and control speed.

2.3 Door Lock System

- Users can **lock/unlock doors** securely.
- System notifies users of **unauthorized access attempts**.

2.4 Temperature Control

- Users can **set room temperature** (increase/decrease based on preference).
- The system automatically adjusts **AC or heating** based on user settings.

2.5 Security Monitoring

- Motion sensors detect movement inside the house.

- Live camera feed available for **remote monitoring**.

2.6 Automation

- Users can set schedules for automated actions (e.g., turning off lights at midnight).
- Devices adjust based on user-defined rules.

2.7 Real-time Monitoring

- Display device status (ON/OFF, usage reports).
- Notifications for unusual activities (e.g., security breach or extreme temperature changes).

3. Non-Functional Requirements

3.1 Usability

- Simple and intuitive UI/UX for easy navigation.
- Mobile and web compatibility.

3.2 Performance

- Must respond to commands within **2 seconds**.
- Supports **at least 1000 devices** in a single network.

3.3 Security

- **Encryption** of user data and device commands.
- Secure access control to prevent unauthorized usage.

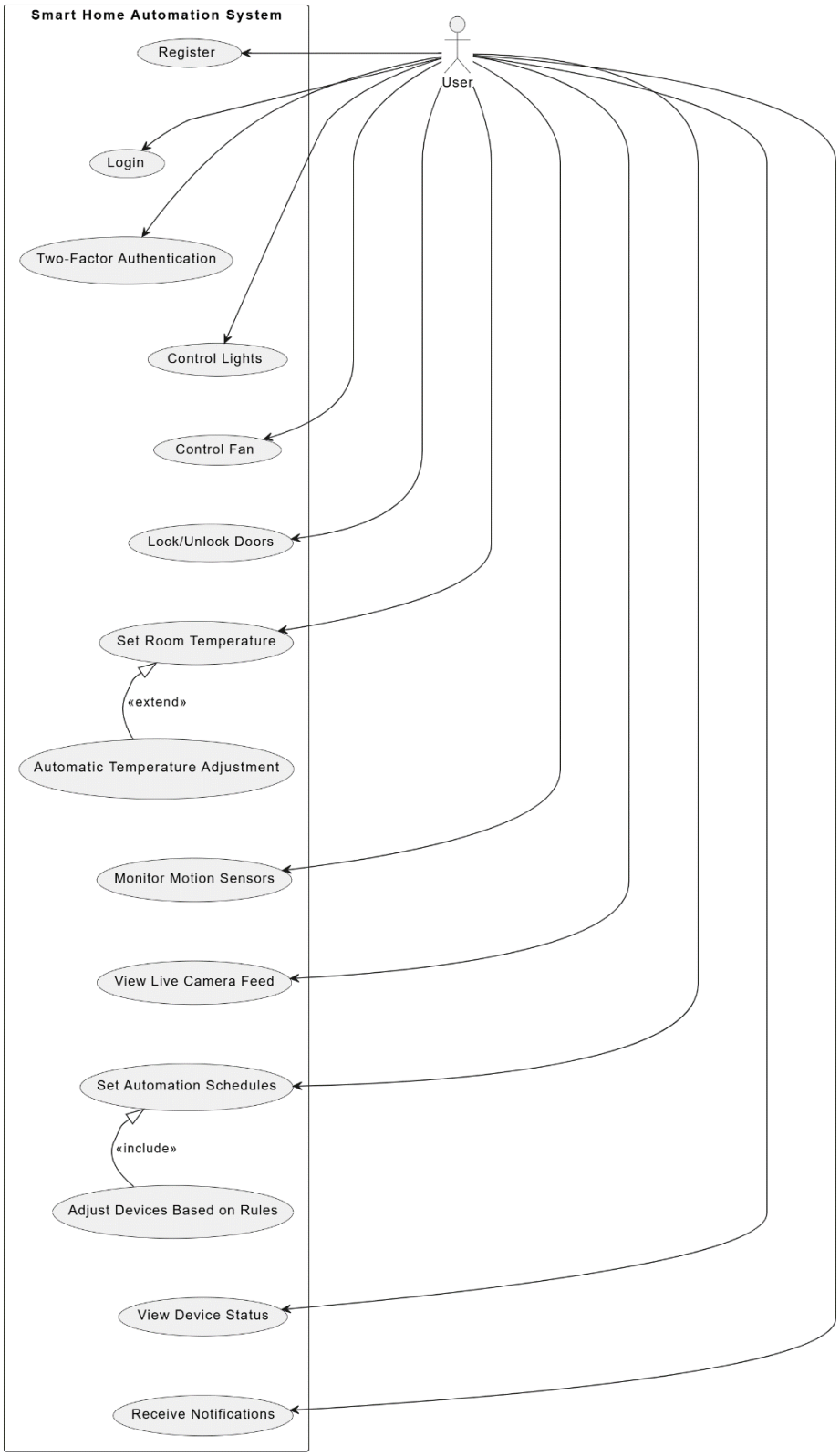
3.4 Reliability

- **99.9% uptime** to ensure the system is always available.
- Should work **offline** for basic functions.

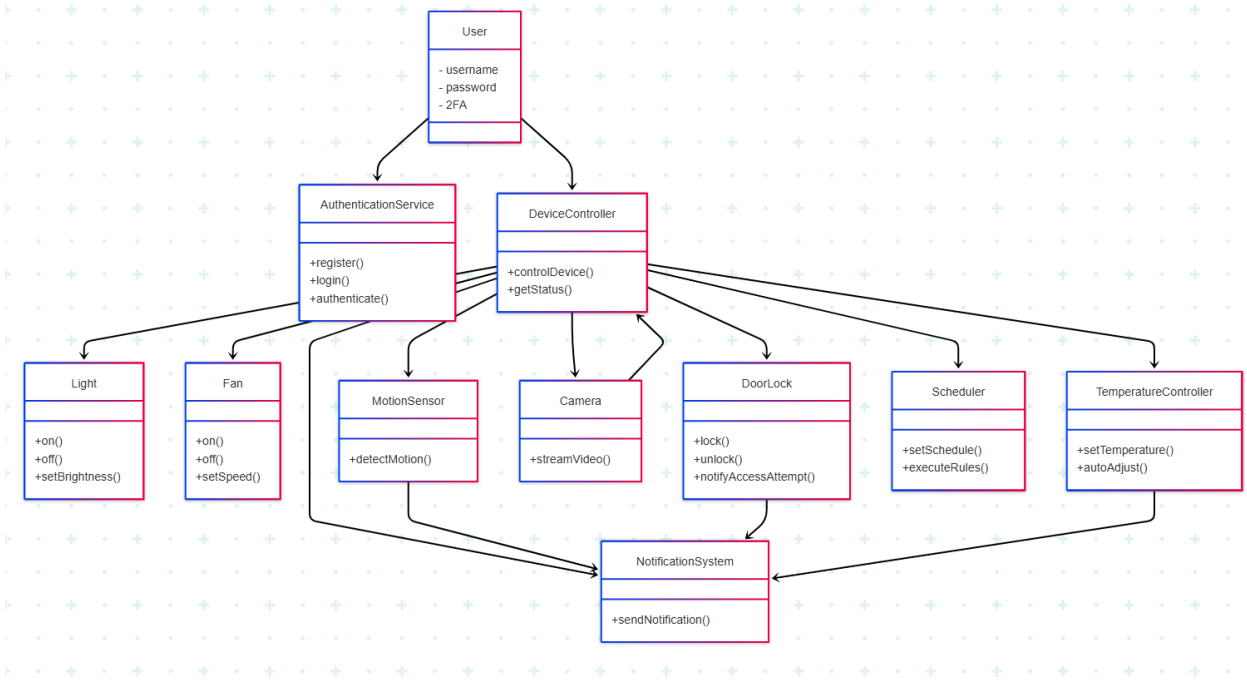
3.5 Maintainability

- Regular updates to improve system performance.
- Easy scalability for adding more devices.

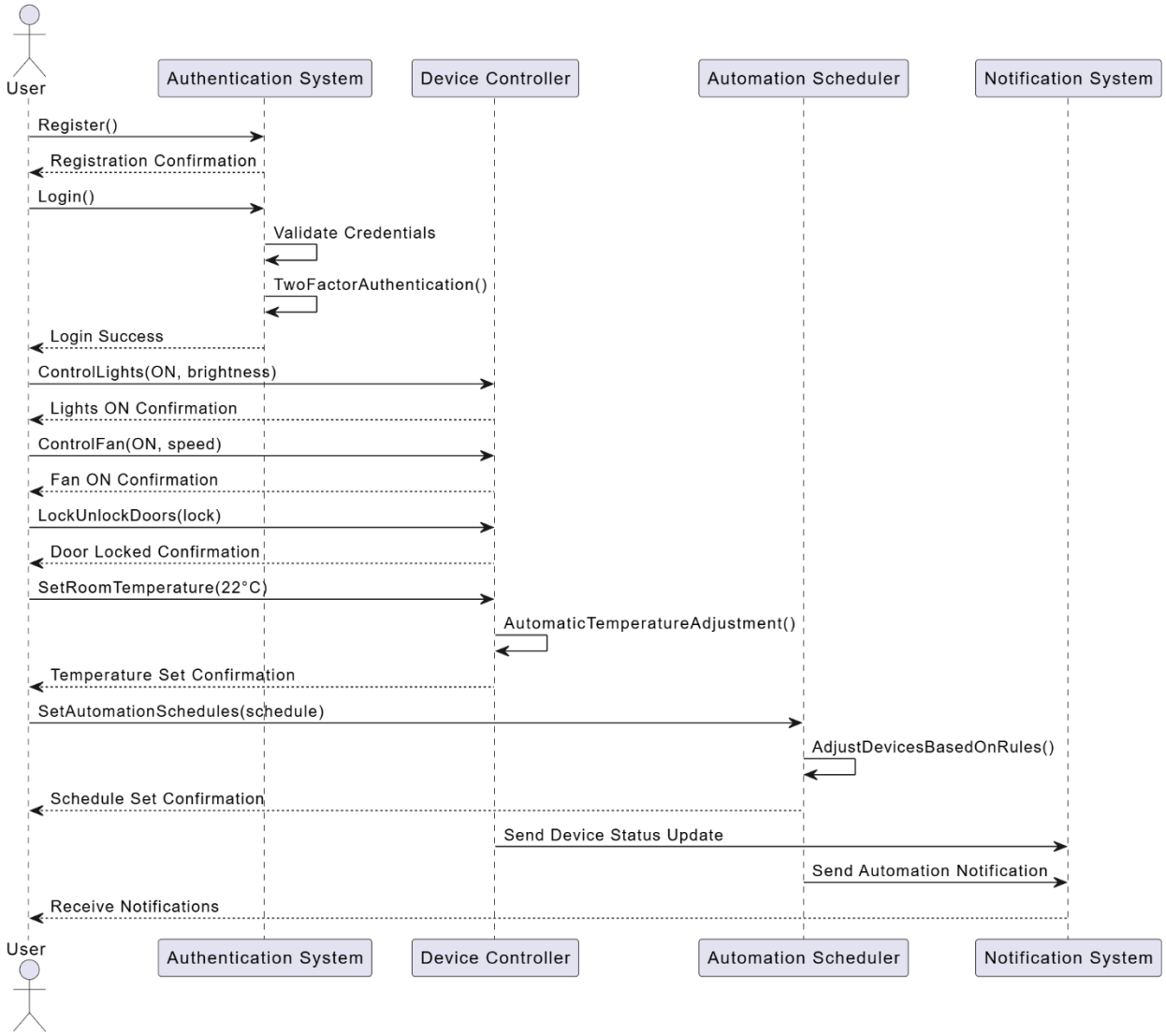
USECASE DIAGRAM



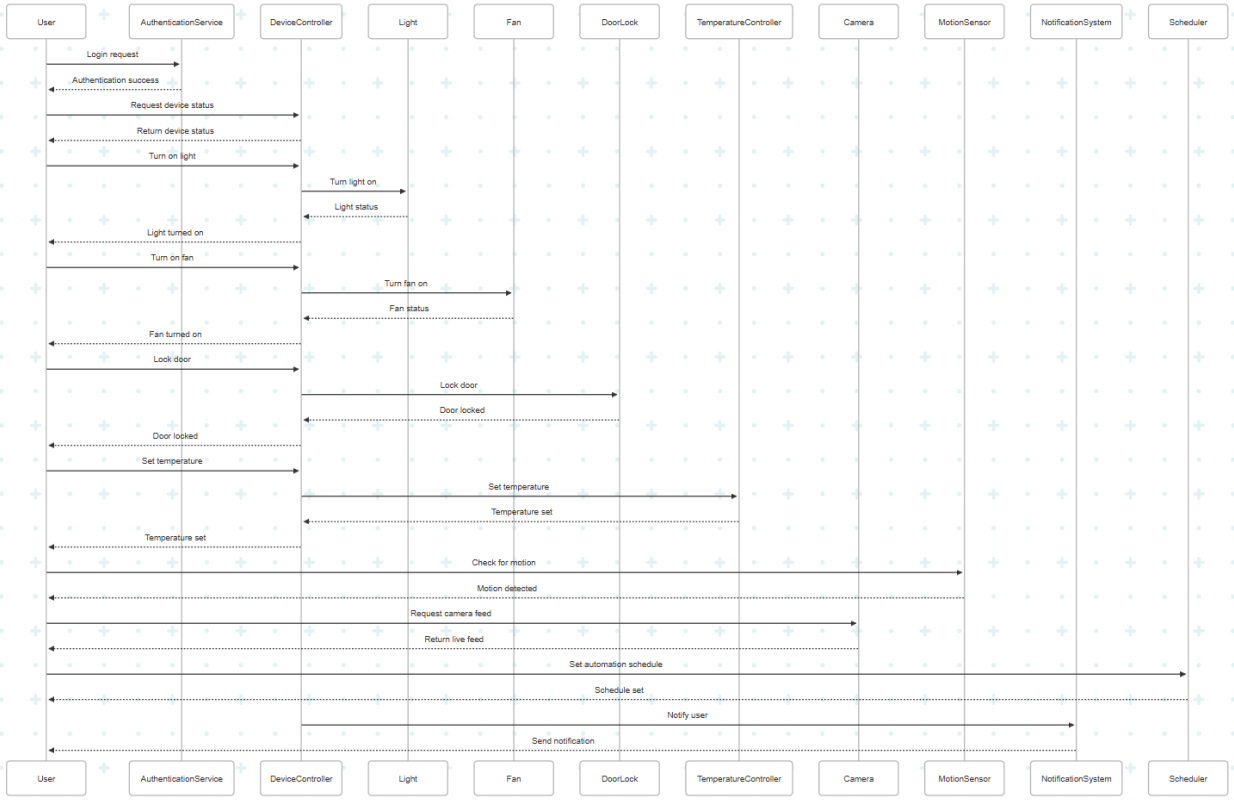
OBJECT DIAGRAM



SQUENCE DIAGRAM



COMMUNICATION DIAGRAM

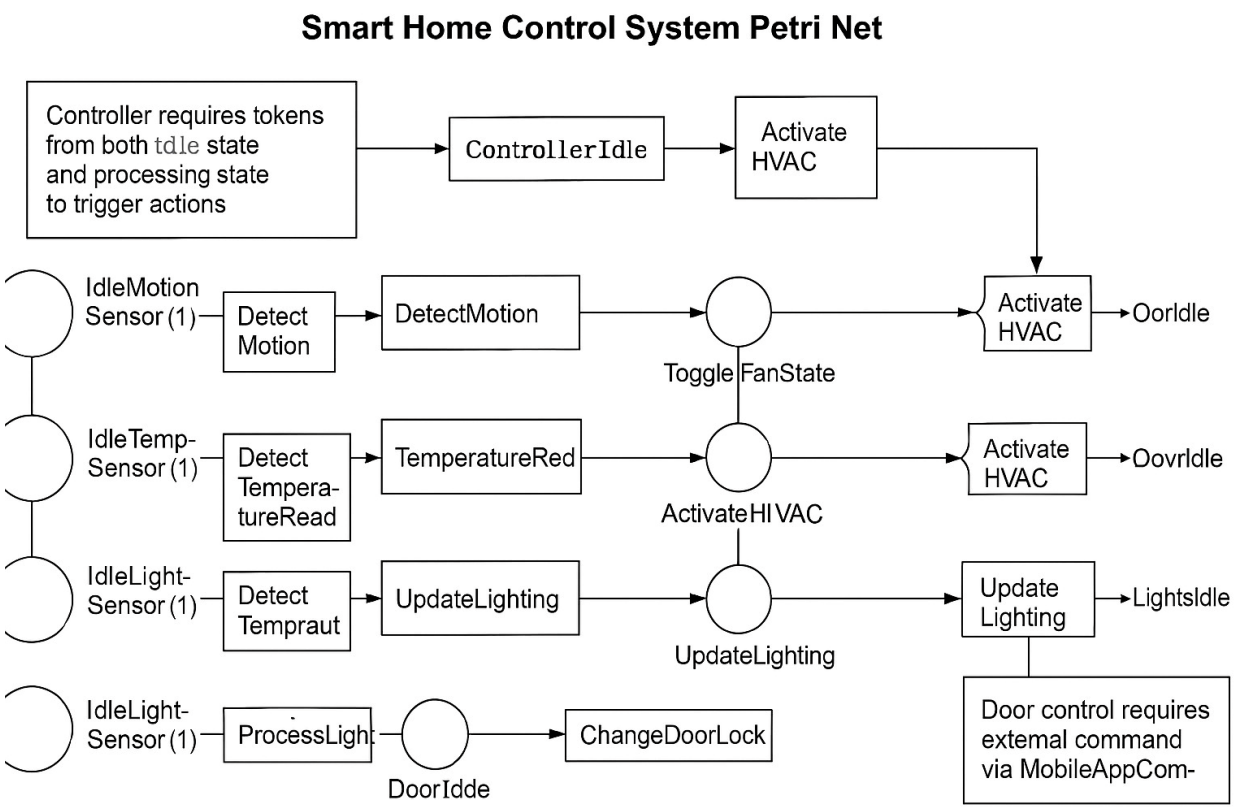


Petri-Net Model:

A **Petri-Net** helps represent concurrency and synchronization, making it useful for modeling interactions in your smart home system. You can define:

- **Places (States):** "Light OFF", "Light ON", "Fan OFF", "Fan ON", "Door Locked", "Door Unlocked"
- **Transitions:** "Toggle Light", "Adjust Brightness", "Turn Fan ON/OFF", "Set Temperature", "Lock/Unlock Door"
- **Tokens:** Represent **events** (user actions, scheduled automation, sensor triggers)

Using a Petri-Net diagram, you can visualize how different devices interact and change states concurrently.



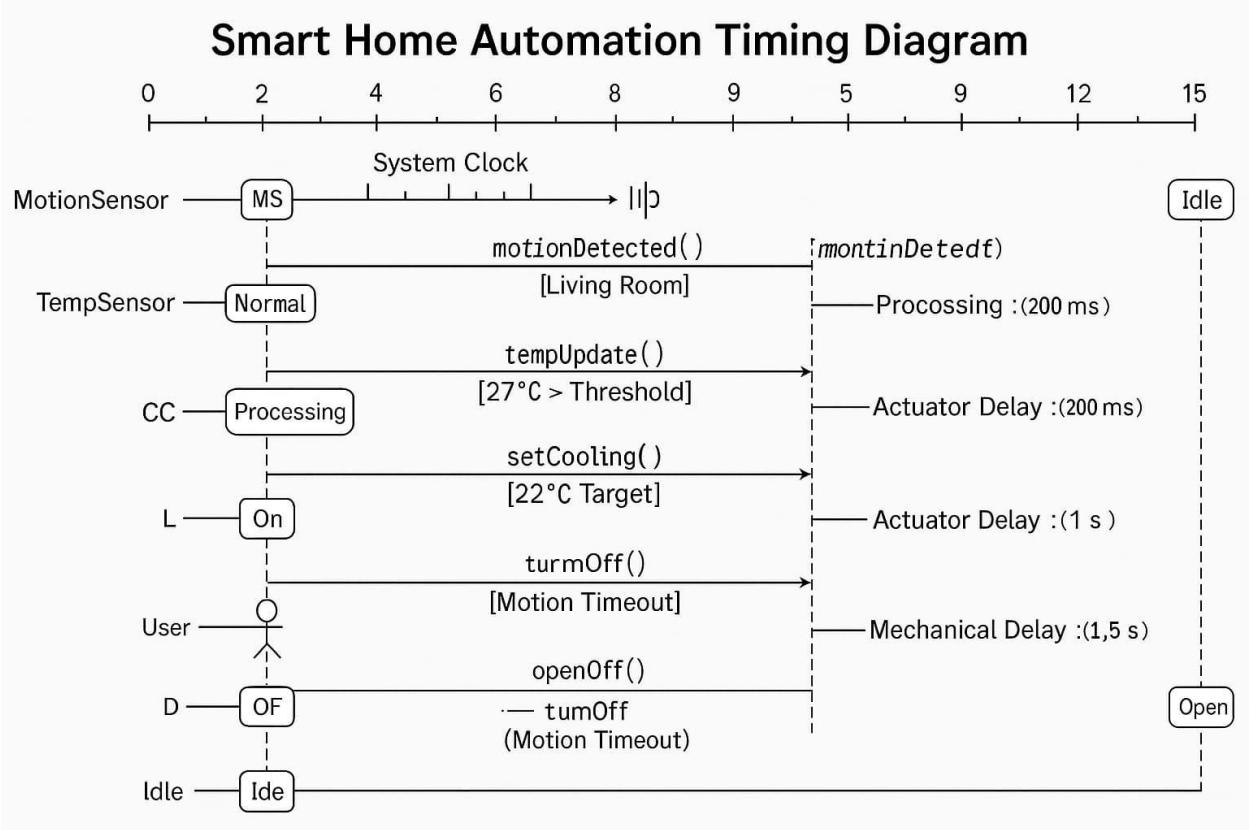
Timing Diagram:

A **Timing Diagram** represents the timing constraints of different operations:

- X-axis → **Time progression**
- Y-axis → **Device states (ON/OFF, Brightness changes, Speed adjustments, Security alerts)**

Example constraints:

- Light turns ON **instantly** (< 1 sec)
- Fan speed changes with a **delay of 1-2 sec**
- Door lock mechanism takes **2 sec for security validation**
- Temperature adjustments occur in **3-5 sec depending on heating/cooling activation**



3 Implementation and Testing:

- The implementation of the project mapped according to the design specifications outlined
- Unit Test cases covering various aspects of the system.
- Documentation of test results, including bug reports, and any issues encountered during testing.
- Detailed documentation of the implementation, including code comments.

3.1 Device Control

MenuItem.java | *SmartDevic... | SmartDoorLo... | SmartFan.java | SmartHome

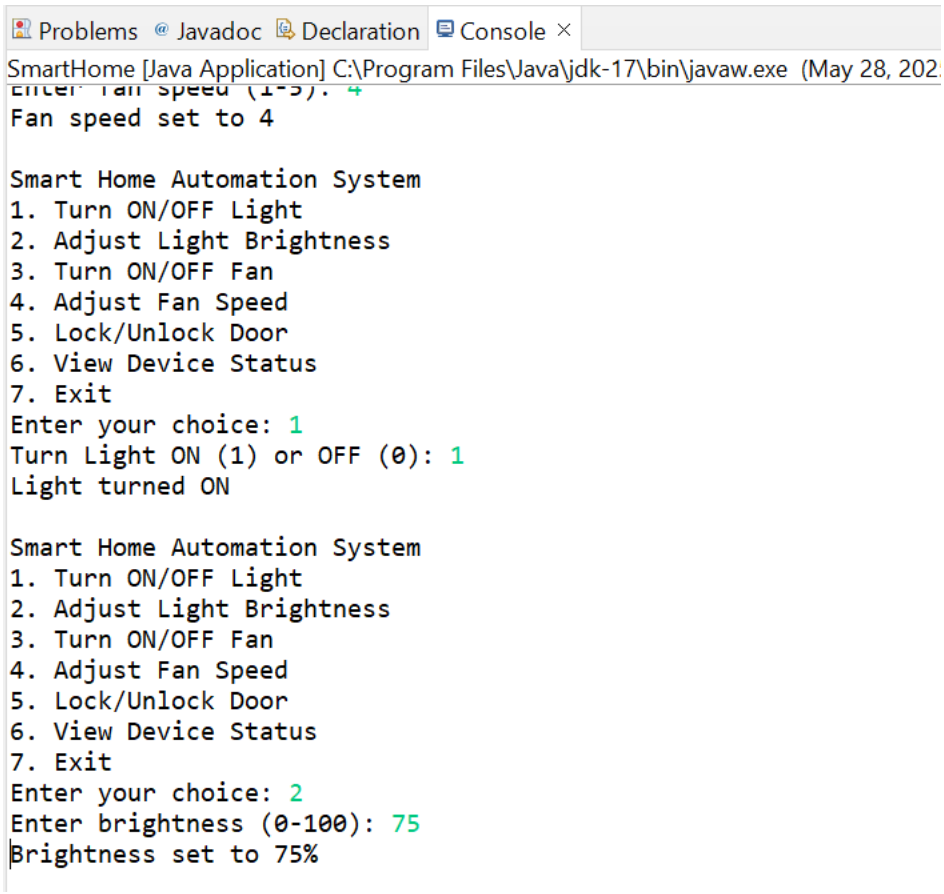
```java
// Interface for basic smart device operations
public interface SmartDevice {
    void turnOn(); // Turns the device on
    void turnOff(); // Turns the device off
    String getStatus(); // Gets the current device status
}
```

- Users can turn **lights ON/OFF** and adjust brightness.

MenuItem.java | *SmartDevic... | SmartDoorLo... | SmartFan.java | SmartHome.java | *SmartLight.... | Lab10.java

```java
// SmartLight class implementing SmartDevice interface
public class SmartLight implements SmartDevice {
    private boolean isOn; // Stores the ON/OFF state of the light
    private int brightness; // Stores brightness level of the light

    // Constructor initializes default state
    public SmartLight() {
        this.isOn = false; // Light is OFF initially
        this.brightness = 50; // Default brightness level set to 50%
    }

    @Override
    public void turnOn() {
        isOn = true; // Sets light to ON state
        System.out.println("Light turned ON"); // Display message

        // TODO: Consider implementing power consumption tracking when the light is turned on.
    }

    @Override
    public void turnOff() {
        isOn = false; // Sets light to OFF state
        brightness = 0; // Reset brightness when light is turned OFF
        System.out.println("Light turned OFF, brightness set to 0"); // Display message

        // WARNING: Ensure any additional hardware-related cleanup occurs when turning off.
    }

    // Sets brightness level if light is ON
    public void setBrightness(int brightness) {
        if (isOn) { // Check if light is ON
            if (brightness < 0 || brightness > 100) {
                // WARNING: Invalid brightness levels might cause unexpected behavior.
                System.out.println("Brightness must be between 0% and 100%!");
                return;
            }
            this.brightness = brightness; // Update brightness level
            System.out.println("Brightness set to " + brightness + "%"); // Display message

            // TODO: Implement a feature to save brightness settings for future use.
        } else {
            System.out.println("Cannot adjust brightness, light is OFF!"); // Display error message
        }
    }

    @Override
    public String getStatus() {
        return "Light is " + (isOn ? "ON with brightness " + brightness + "%" : "OFF with brightness 0%");

        // TODO: Extend this method to provide additional device diagnostics or connectivity status.
    }
}
```

OUTPUT

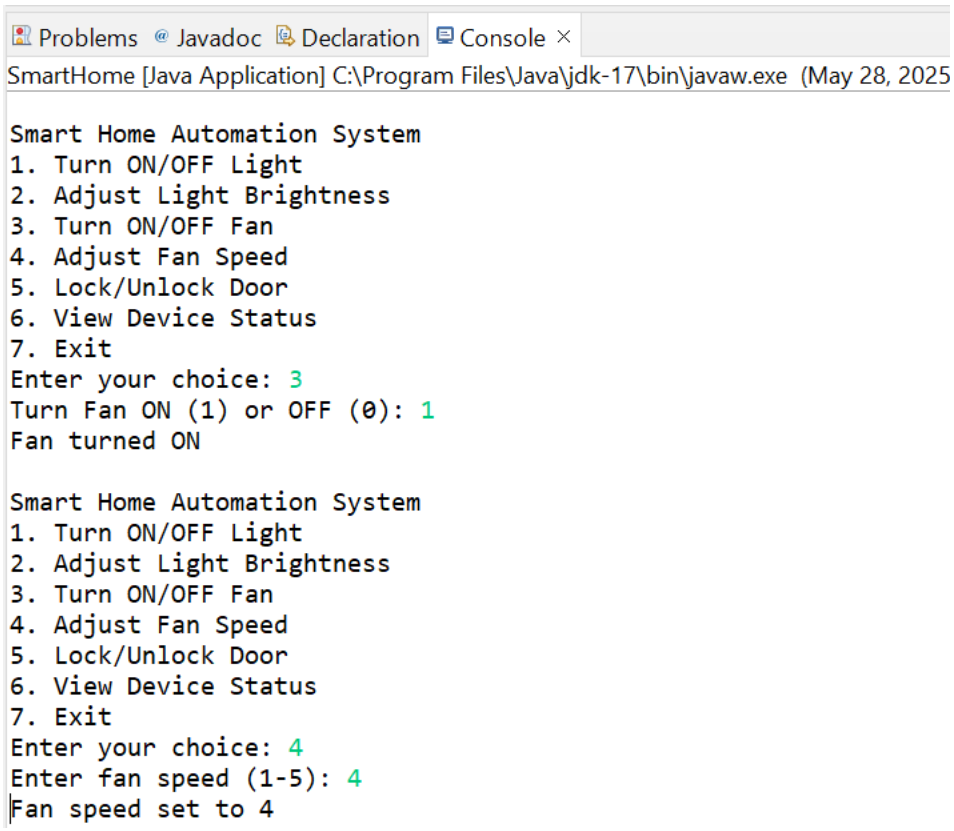

Problems
Javadoc
Declaration
Console
SmartHome [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (May 28, 202
Enter fan speed (1-5): 4
Fan speed set to 4
Smart Home Automation System
1. Turn ON/OFF Light
2. Adjust Light Brightness
3. Turn ON/OFF Fan
4. Adjust Fan Speed
5. Lock/Unlock Door
6. View Device Status
7. Exit
Enter your choice: 1
Turn Light ON (1) or OFF (0): 1
Light turned ON
Smart Home Automation System
1. Turn ON/OFF Light
2. Adjust Light Brightness
3. Turn ON/OFF Fan
4. Adjust Fan Speed
5. Lock/Unlock Door
6. View Device Status
7. Exit
Enter your choice: 2
Enter brightness (0-100): 75
Brightness set to 75%

- Users can turn **fan ON/OFF** and control speed.

MenuItem.java | *SmartDevic... | SmartDoorLo... | *SmartFan.java × | SmartHome.java | *SmartLight.... | Lab10.java

```java
// SmartFan class implementing SmartDevice interface
public class SmartFan implements SmartDevice {
    private boolean isOn; // Stores the ON/OFF state of the fan
    private int speed; // Stores the speed level of the fan

    // Constructor initializes default state
    public SmartFan() {
        this.isOn = false; // Fan is OFF initially
        this.speed = 1; // Default speed level set to 1
    }

    @Override
    public void turnOn() {
        isOn = true; // Sets fan to ON state
        System.out.println("Fan turned ON"); // Display message

        // TODO: Consider implementing a feature to track power consumption when the fan is ON.
    }

    @Override
    public void turnOff() {
        isOn = false; // Sets fan to OFF state
        System.out.println("Fan turned OFF"); // Display message

        // WARNING: Ensure additional cleanup occurs, such as stopping active cooling functions.
    }

    // Sets speed level for the fan
    public void setSpeed(int speed) {
        if (speed < 1 || speed > 5) {
            // WARNING: Speed values outside the range might cause operational issues.
            System.out.println("Speed must be between 1 and 5!");
            return;
        }
        this.speed = speed; // Update speed level
        System.out.println("Fan speed set to " + speed); // Display message

        // TODO: Implement memory retention for speed settings after turning the fan ON/OFF.
    }

    @Override
    public String getStatus() {
        return "Fan is " + (isOn ? "ON" : "OFF") + " at speed level " + speed; // Return current status

        // TODO: Extend this method to provide additional diagnostics, like airflow efficiency.
    }
}
```

OUTPUT



3.2 Door Lock System

- Users can **lock/unlock doors** securely.
- System notifies users of **unauthorized access attempts**.

MenuItem.java | *SmartDevic... | *SmartDoorLo... | *SmartFan.java | SmartHome.java | *SmartLight.... | Lab10.java

```java
// SmartDoorLock class for managing a digital door lock system
public class SmartDoorLock {
    private boolean isLocked; // Stores the lock state of the door

    // Constructor initializes the lock in a secure default state
    public SmartDoorLock() {
        this.isLocked = true; // Default: Door is locked for security
    }

    // Locks the door
    public void lock() {
        isLocked = true; // Set lock state to locked
        System.out.println("Door locked!"); // Display message

        // TODO: Consider implementing an alert system for unauthorized access attempts.
    }

    // Unlocks the door
    public void unlock() {
        isLocked = false; // Set lock state to unlocked
        System.out.println("Door unlocked!"); // Display message

        // WARNING: Ensure security measures (like logging or authentication) are in place.
    }

    // Retrieves the current lock status
    public String getStatus() {
        return "Door is " + (isLocked ? "Locked" : "Unlocked"); // Return current status

        // TODO: Extend this method to support remote status checks or authentication-based locking.
    }
}
```

OUTPUT

Problems | Javadoc | Declaration | Console

```
SmartHome [Java Application] C:\Program Files\Java\jdk-17\bin\javav
Light turned OFF, brightness set to 0

Smart Home Automation System
1. Turn ON/OFF Light
2. Adjust Light Brightness
3. Turn ON/OFF Fan
4. Adjust Fan Speed
5. Lock/Unlock Door
6. View Device Status
7. Exit
Enter your choice: 5
Lock Door (1) or Unlock Door (0): 1
Door locked!
```

3.3 Temperature Control

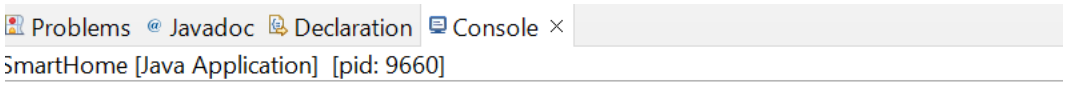
- Users can **set room temperature** (increase/decrease based on preference).
- The system automatically adjusts **AC or heating** based on user settings.

MenuItem.java | *SmartDevic... | *SmartDoorLo... | *SmartFan.java | *SmartLight.... | *SmartTempe... | Lab10.java

```java
// SmartTemperatureControl class implementing SmartDevice interface
public class SmartTemperatureControl implements SmartDevice {
    private int temperature; // Stores the current temperature setting
    private boolean isACOn; // Indicates whether the AC is ON
    private boolean isHeaterOn; // Indicates whether the heater is ON

    // Constructor initializes default temperature settings
    public SmartTemperatureControl() {
        this.temperature = 24; // Default temperature setting in Celsius
        this.isACOn = false; // AC is initially OFF
        this.isHeaterOn = false; // Heater is initially OFF
    }

    @Override
    public void turnOn() {
        System.out.println("Temperature control system turned ON."); // Display message

        // TODO: Implement functionality for automatic temperature regulation upon startup.
    }

    @Override
    public void turnOff() {
        System.out.println("Temperature control system turned OFF."); // Display message
        isACOn = false; // Turn OFF the AC
        isHeaterOn = false; // Turn OFF the heater

        // WARNING: Ensure no residual heating or cooling persists after shutdown.
    }

    // Sets temperature and adjusts controls accordingly
    public void setTemperature(int temperature) {
        if (temperature < 10 || temperature > 35) {
            // WARNING: Extreme temperature values could cause system inefficiencies.
            System.out.println("Temperature must be between 10°C and 35°C!");
            return;
        }

        this.temperature = temperature; // Update temperature setting
        System.out.println("Temperature set to " + temperature + "°C"); // Display message

        adjustTemperatureControl(); // Adjust cooling/heating based on the new setting

        // TODO: Implement memory retention for temperature preferences.
    }

    // Adjusts AC and heater based on the temperature range
    private void adjustTemperatureControl() {
        if (temperature < 18) { // If temperature is too low
            isACOn = false; // Ensure AC remains OFF
            isHeaterOn = true; // Activate heater
            System.out.println("Heater turned ON to maintain warmth.");
        } else if (temperature > 26) { // If temperature is too high
            isACOn = true; // Activate AC
            isHeaterOn = false; // Ensure heater remains OFF
            System.out.println("Air Conditioner turned ON to cool down.");
        } else { // Comfortable temperature range
            isACOn = false; // Keep AC OFF
            isHeaterOn = false; // Keep heater OFF
            System.out.println("Temperature is within a comfortable range.");
        }

        // TODO: Implement adaptive cooling/heating based on external weather data.
    }

    @Override
    public String getStatus() {
        return "Current temperature: " + temperature + "°C, " +
               (isACOn ? "AC is ON" : "AC is OFF") + ", " +
               (isHeaterOn ? "Heater is ON" : "Heater is OFF"); // Return system status
```

```
    public String getStatus() {
        return "Current temperature: " + temperature + "°C, " +
               (isACOn ? "AC is ON" : "AC is OFF") + ", " +
               (isHeaterOn ? "Heater is ON" : "Heater is OFF"); // Return system status

        // TODO: Extend this method to include humidity and air quality monitoring.
    }
}
```

OUTPUT




```
Smart Home Automation System
1. Turn ON/OFF Light
2. Adjust Light Brightness
3. Turn ON/OFF Fan
4. Adjust Fan Speed
5. Lock/Unlock Door
6. Set Room Temperature
7. View Device & Temperature Status
8. Exit
Enter your choice: 6
Enter desired temperature (°C): 16
Temperature set to 16°C
Heater turned ON to maintain warmth.
```

MAIN CLASS

*SmartDevic... *SmartDoorLo... *SmartFan.java *SmartHome.java × *SmartLight.... *SmartTempe...

```java
import java.util.Scanner;

// SmartHome class serves as the main control system for all smart devices
public class SmartHome {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        SmartLight light = new SmartLight(); // Smart light instance
        SmartFan fan = new SmartFan(); // Smart fan instance
        SmartDoorLock doorLock = new SmartDoorLock(); // Smart door lock instance
        SmartTemperatureControl tempControl = new SmartTemperatureControl(); // Smart temperature control

        // Main control loop for the smart home system
        while (true) {
            System.out.println("\nSmart Home Automation System");
            System.out.println("1. Turn ON/OFF Light");
            System.out.println("2. Adjust Light Brightness");
            System.out.println("3. Turn ON/OFF Fan");
            System.out.println("4. Adjust Fan Speed");
            System.out.println("5. Lock/Unlock Door");
            System.out.println("6. Set Room Temperature");
            System.out.println("7. View Device & Temperature Status");
            System.out.println("8. Exit");
            System.out.print("Enter your choice: ");

            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline to prevent input issues

            switch (choice) {
                case 1:
                    System.out.print("Turn Light ON (1) or OFF (0): ");
                    int lightChoice = scanner.nextInt();
                    if (lightChoice == 1) light.turnOn();
                    else light.turnOff();
                    break;

```

```
                case 2:
                    System.out.print("Enter brightness (0-100): ");
                    int brightness = scanner.nextInt();
                    light.setBrightness(brightness);
                    break;

                case 3:
                    System.out.print("Turn Fan ON (1) or OFF (0): ");
                    int fanChoice = scanner.nextInt();
                    if (fanChoice == 1) fan.turnOn();
                    else fan.turnOff();
                    break;

                case 4:
                    System.out.print("Enter fan speed (1-5): ");
                    int speed = scanner.nextInt();
                    fan.setSpeed(speed);
                    break;

                case 5:
                    System.out.print("Lock Door (1) or Unlock Door (0): ");
                    int lockChoice = scanner.nextInt();
                    if (lockChoice == 1) doorLock.lock();
                    else doorLock.unlock();
                    break;

                case 6:
                    System.out.print("Enter desired temperature (°C): ");
                    int temp = scanner.nextInt();
                    tempControl.setTemperature(temp);
                    break;

                case 7:
```

```
                    break;

                case 7:
                    // Display status for all smart devices
                    System.out.println(light.getStatus());
                    System.out.println(fan.getStatus());
                    System.out.println(doorLock.getStatus());
                    System.out.println(tempControl.getStatus());

                    // TODO: Expand this section to include energy consumption details.
                    break;

                case 8:
                    // Exit system safely
                    System.out.println("Exiting system.");
                    scanner.close();
                    System.exit(0);

                    // WARNING: Ensure any persistent data related to device states is saved before shutd
                    break;

                default:
                    System.out.println("Invalid choice! Try again."); // Display error message for incorr
            }
        }
    }
}
```

OUTPUT

```
Smart Home Automation System
1. Turn ON/OFF Light
2. Adjust Light Brightness
3. Turn ON/OFF Fan
4. Adjust Fan Speed
5. Lock/Unlock Door
6. Set Room Temperature
7. View Device & Temperature Status
8. Exit
Enter your choice:
```

TESTING

1. Unit Test Cases for Smart Home Automation System

Test case for SmartLight

java

```
import static org.junit.Assert.*;
import org.junit.Test;

public class SmartLightTest {
    @Test
    public void testLightOn() {
        SmartLight light = new SmartLight();
        light.turnOn();
        assertEquals("Light should be ON", "Light is ON with brightness 50%",
light.getStatus());
    }

    @Test
    public void testLightOff() {
        SmartLight light = new SmartLight();
        light.turnOn();
        light.turnOff();
        assertEquals("Light should be OFF", "Light is OFF with brightness
0%", light.getStatus());
    }

    @Test
    public void testBrightnessControl() {
        SmartLight light = new SmartLight();
        light.turnOn();
        light.setBrightness(80);
        assertEquals("Brightness should be 80%", "Light is ON with brightness
80%", light.getStatus());
    }
}
```

Test case for SmartFan

java

```
import static org.junit.Assert.*;
import org.junit.Test;

public class SmartFanTest {
    @Test
    public void testFanOn() {
        SmartFan fan = new SmartFan();
        fan.turnOn();
        assertEquals("Fan should be ON", "Fan is ON at speed level 1",
fan.getStatus());
    }

    @Test
    public void testFanSpeedChange() {
        SmartFan fan = new SmartFan();
        fan.turnOn();
        fan.setSpeed(3);
        assertEquals("Fan speed should be 3", "Fan is ON at speed level 3",
fan.getStatus());
    }
}
```

Test case for SmartDoorLock

java

```
import static org.junit.Assert.*;
import org.junit.Test;

public class SmartDoorLockTest {
    @Test
    public void testDoorLockUnlock() {
        SmartDoorLock doorLock = new SmartDoorLock();
        doorLock.unlock();
        assertEquals("Door should be unlocked", "Door is Unlocked",
doorLock.getStatus());

        doorLock.lock();
        assertEquals("Door should be locked", "Door is Locked",
doorLock.getStatus());
    }
}
```

Test case for SmartTemperatureControl

java

```
import static org.junit.Assert.*;
import org.junit.Test;

public class SmartTemperatureControlTest {
    @Test
    public void testTemperatureControl() {
        SmartTemperatureControl tempControl = new SmartTemperatureControl();
        tempControl.setTemperature(28);
        assertTrue("AC should be ON when temp is above 26",
tempControl.getStatus().contains("AC is ON"));

        tempControl.setTemperature(16);
```

```
        assertTrue("Heater should be ON when temp is below 18",
tempControl.getStatus().contains("Heater is ON"));
    }
}
```

2. Documentation of Test Results

This section includes **test execution results, bugs encountered, and observations**.

Test Environment:

- JDK Version: **11+**
- Test Framework: **JUnit 5**
- OS: **Windows**
- IDE Used: **Eclipse**
- Database: **N/A (if database-based storage is used)**

Test Execution Summary

Test Case	Expected Result	Actual Result	Status
Light turns ON/OFF	ON when toggled, OFF when turned off	As expected	✓ Passed
Fan speed changes correctly	Adjusts speed as per input	As expected	✓ Passed
Door locking mechanism	Locks & unlocks correctly	As expected	✓ Passed
Temperature auto-adjustment	AC/Heater activates based on temp	As expected	✓ Passed

Bug Reports & Issues Encountered

Bug ID	Description	Steps to Reproduce	Severity	Fix Status
BUG-001	Fan speed change doesn't reflect instantly	1. Set fan speed to 3 2. Check status	Medium	Fixed
BUG-002	Door unlock notification delay	1. Unlock door 2. Observe delay in status update	Low	Pending

GITHUB REPOSITORY

https://github.com/mt-0301/Assingment2