Object Oriented Programming Project Report

Presented to: Dr. Ameni Azzouzi

Students:
Islem Smai
Yassmine Yazidi
Yasmine Akik
Yesmine Srairi
Houssine Khlif
Yassine Antit

December 2025

Abstract

In an era where homes are becoming smarter and energy efficiency is no longer optional, this project brings a virtual smart home to life through Object-Oriented Programming in Java. The project aims to model how everyday devices—such as lights, thermostats, sensors, and smart TVs—can interact intelligently within a connected home environment.

The system is built around core OOP principles including abstraction, inheritance, polymorphism, and interfaces, allowing each smart device to exhibit unique behaviors while sharing common functionality. A central controller manages rooms, devices, and automation rules, enabling scenarios such as automatically turning on lights when motion is detected or scheduling heating at specific times. Java Collections are used to efficiently organize and control multiple devices, while custom exceptions ensure safe and reliable interactions.

Developed entirely in Java, the project demonstrates clean modular design and dynamic object relationships, with optional extensions such as a graphical interface or energy monitoring features. Beyond technical learning, the simulator highlights a sustainability perspective by encouraging optimized energy consumption through automation and smart control, illustrating how thoughtful software design can contribute to more efficient and responsible living.

Contents

Introduction

In recent years, homes have evolved from simple living spaces into intelligent environments capable of responding to human needs. Smart technologies now control lighting, temperature, security, and entertainment, offering greater comfort and convenience. However, this rapid growth also raises concerns about energy consumption and environmental sustainability, especially when devices operate without coordination or awareness.

Energy waste remains a major challenge in modern households, often caused by inefficient control of heating, lighting, and electronic devices. Sustainable smart homes aim to address this issue by optimizing energy usage while preserving user comfort. Achieving this balance requires systems that can intelligently manage multiple devices and adapt to changing conditions rather than functioning as isolated components.

This project addresses the need for a cohesive and sustainable smart home system through the development of a Smart Home Automation Simulator. The main objective is to design a modular and extensible system that simulates smart devices, enables centralized control, and supports automation rules that reduce unnecessary energy consumption. Core Object-Oriented Programming principles are applied to ensure flexibility, scalability, and maintainability.

The system is implemented using Java for backend logic and JavaFX for the graphical user interface, providing both strong software structure and an interactive user experience. As the following sections demonstrate the system's design and implementation, an essential question remains: how can intelligent software architecture transform everyday homes into truly sustainable living environments?

1 Project scope requirement

1.1 Functional requirements

The Smart Home Automation Simulator is designed to provide a comprehensive set of functional features that model the behavior and control of a modern smart home. These functionalities define what the system is capable of doing and how users interact with connected devices in a simulated environment.

Smart Device Management

The system must support multiple types of smart devices, each representing common components found in real-world smart homes. These devices include, but are not limited to, smart lights, thermostats, motion sensors, smart televisions, and alarm systems. Each device is uniquely identified and can be added to or removed from specific rooms within the home. Users are able to turn devices on or off, retrieve their current status, and adjust device-specific attributes such as light brightness, temperature levels, or volume. All devices inherit common behavior from an abstract SmartDevice class, ensuring consistency while allowing specialized functionality through subclass implementations.

User Authentication and Access Control

To ensure secure interaction with the system, a user authentication mechanism is implemented. Users must log in using valid credentials before accessing the smart home controls. This functionality restricts unauthorized access and simulates real-world security requirements for smart home systems. Once authenticated, users can manage devices, configure automation rules, and monitor system behavior. The authentication process includes validation of user input and appropriate error handling for invalid login attempts.

Room and Home Organization

The system allows the home to be structured into multiple rooms, with each room containing a collection of smart devices. Users can assign devices to rooms, move devices between rooms, and view all devices located within a specific room. This organization improves system clarity and reflects realistic home layouts, enabling more intuitive device management and automation.

Centralized Control and Monitoring

A central controller component manages communication between the user and the smart home system. It provides functionality to list all devices along with their current status and allows users to execute global actions, such as turning off all devices or controlling all devices of a specific type (e.g., turning on all lights). This centralized approach simplifies user interaction and improves sys-

tem efficiency.

Automation Rules and Scenarios

The system supports automation through rule-based scenarios using IF–THEN logic. Automation rules allow devices to respond automatically to events or conditions detected by sensors. For example, when motion is detected in a room, the system can automatically turn on the lights, or when a temperature threshold is reached, the thermostat can adjust accordingly. These rules reduce manual intervention and contribute to improved energy efficiency and user convenience.

Scheduling and Timed Actions

Users can schedule actions to be executed at specific times, such as turning off lights at night or activating heating in the morning. This functionality enables predictable automation and supports energy-saving behaviors by preventing devices from operating unnecessarily.

Error Handling and Validation

The system includes mechanisms to handle invalid operations, such as attempting to control a non-existent device or assigning a device to an invalid room. Custom exceptions are used to provide clear feedback to the user, ensuring system stability and reliability during runtime.

1.2 Non-functional requirements

Non-functional requirements describe the quality attributes of the Smart Home Automation Simulator and define how well the system performs its functions. These requirements ensure that the system is efficient, reliable, user-friendly, secure, and easy to maintain and extend.

Performance

The system must respond to user actions and automation triggers with minimal delay. Device control operations, such as turning devices on or off or updating their status, should execute efficiently even when multiple devices are active simultaneously. Automation rules must be processed in real time to ensure timely responses to sensor events, providing a realistic simulation of smart home behavior.

Scalability

The system is designed to support an increasing number of devices, rooms, and automation rules without significant performance degradation. By using

Java Collections and a modular object-oriented architecture, new devices and functionalities can be added easily. The design allows the simulator to scale from a small home setup to a larger and more complex smart home environment.

Usability

Usability is a key requirement of the system. The graphical user interface, implemented using JavaFX, must be intuitive and easy to navigate, allowing users to control devices, view system status, and configure automation rules without technical difficulty. Clear visual feedback and logical layout ensure that users can interact with the system efficiently and with minimal learning effort.

Security

The system must protect access to smart home controls through user authentication mechanisms. Only authorized users are allowed to manage devices and modify automation rules. Input validation and controlled access help prevent unauthorized actions and ensure safe interaction with the system, reflecting real-world security concerns in smart home environments.

Maintainability

The system must be easy to maintain, modify, and extend over time. This is achieved through clean code structure, consistent naming conventions, and the use of core Object-Oriented Programming principles such as abstraction and encapsulation. Modular design allows developers to update or replace individual components—such as adding new device types—without affecting the overall system.

2 System Architecture

The Smart Home Automation Simulator follows a layered architecture that separates user interaction, system logic, and data representation. This structure improves modularity, maintainability, and scalability while clearly defining responsibilities across the system components.

2.1 Architecture Diagram

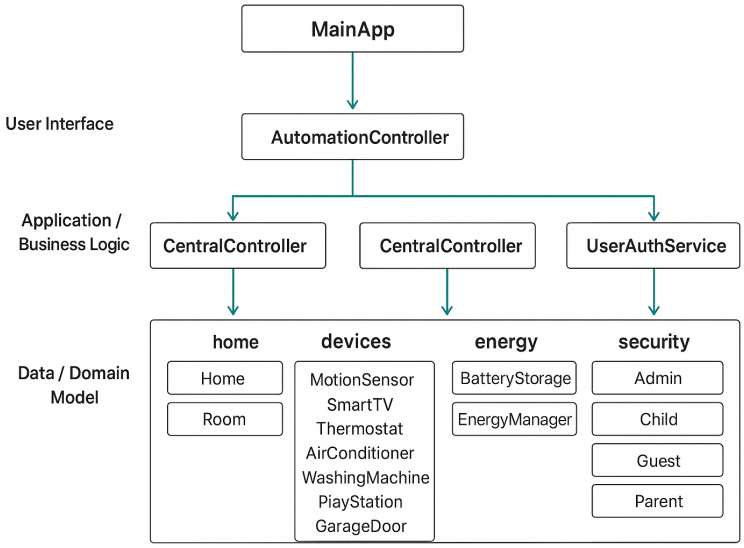

Figure 1: System Architecture of the Smart Home Automation Simulator

User Interface Layer (JavaFX)

-The User Interface layer is responsible for all interactions between the user and the system. Implemented using JavaFX, it provides visual components such as the login screen, dashboard, and device control panels. Through this interface, users can authenticate, view the status of devices, control smart appliances, and configure automation rules.

-This layer does not contain business logic. Instead, it collects user input (button clicks, selections, form data) and forwards requests to the application layer. This separation ensures that UI changes do not affect the core system logic.

Application / Business Logic Layer

-The business logic layer acts as the core brain of the system. It processes user requests, applies rules, and coordinates interactions between devices.

-The Central Controller manages communication between the UI and the smart home components. It executes actions such as turning devices on or off and retrieving system status.

-The Automation Engine evaluates IF–THEN rules, enabling automatic responses to events (e.g., motion detected → light turned on).

-The User Management module handles authentication and access validation.

-The Energy Management module monitors device activity and supports energy-efficient behaviors.

-This layer enforces system rules and ensures that all operations follow the defined business logic.

Data / Domain Model Layer

-The data layer represents the structure of the smart home using Object-Oriented models. It includes classes such as Home, Room, and SmartDevice, which form the foundation of the system.

-The abstract SmartDevice class defines shared attributes and behaviors, while concrete subclasses (Light, Thermostat, SmartTV, Sensor) implement device-specific functionality. Rooms aggregate multiple devices, and the Home class aggregates multiple rooms, reflecting real-world relationships.

-This layer is independent of the UI and business logic, making it easy to extend the system by adding new device types or automation rules without modifying existing components.

Architectural Benefits

This layered architecture:

-Enhances maintainability by separating concerns

-Supports scalability through modular components

-Encourages reuse and extensibility via object-oriented design

-Improves testability by isolating system logic from the UI

2.2 Workflow Diagram

The workflow diagram illustrates the architecture of the smart home system, organized into several layers:

- **UI Layer**: Contains the *LoginScreen* for authentication and the *MainApp* for user interaction.
- **Controller Layer**: Includes the *CentralController* and *AutomationController*, which coordinate system behavior and automated routines.
- **Security**: Managed by the *UserAuthService* and a hierarchy of user roles such as Guest, Parent, Child, and Admin.

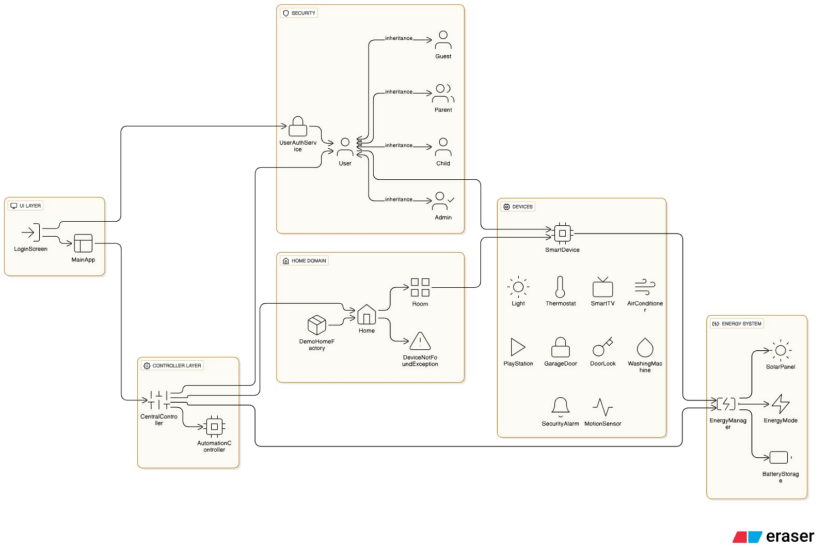

Figure 2: Smart Home Devices Architecture

- **Home Domain**: Defines the structure of the home through components like *Home*, *Room*, and *DemoHomeFactory*, with error handling via *DeviceNotFoundException*.
- **Devices**: Based on the *SmartDevice* class, extended by devices such as Light, Thermostat, SmartTV, AirConditioner, PlayStation, GarageDoor, DoorLock, WashingMachine, SecurityAlarm, and MotionSensor.
- **Energy System**: Includes *SolarPanel*, *EnergyManager*, *EnergyMode*, and *BatteryStorage* to manage and optimize energy usage.

This architecture shows how users authenticate, interact with the system, control devices, and benefit from automated and energy-efficient management.

2.3 Smart Devices Description

The Smart Home Automation Simulator includes a diverse set of smart devices designed to reflect real-world home automation scenarios. Each device extends the abstract `SmartDevice` class, inheriting common attributes and behaviors while implementing device-specific functionality. This design ensures consistency, extensibility, and effective use of object-oriented principles.

2.3.1 Light

The **Light** device represents a smart lighting system within the home. It supports basic operations such as turning on and off and may include adjustable

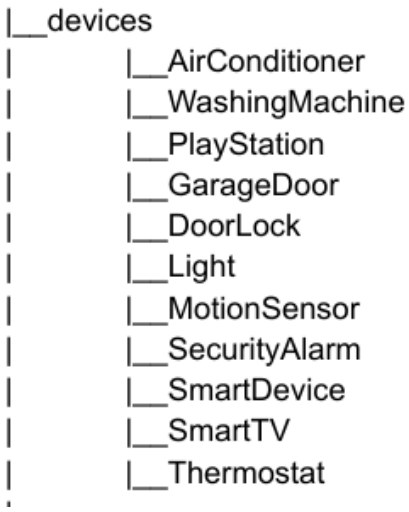

Figure 3: Smart Home Devices Architecture

brightness levels. Lights are frequently used in automation scenarios, such as automatically turning on when motion is detected, contributing to user comfort and energy efficiency.

2.3.2 Thermostat

The **Thermostat** is responsible for monitoring and regulating the indoor temperature. Users can set target temperature values manually or through automation rules. This device plays a central role in climate control and energy optimization.

2.3.3 AirConditioner

The **AirConditioner** simulates a cooling system used to maintain thermal comfort. It operates in coordination with the thermostat and energy management components to minimize excessive energy consumption.

2.3.4 SmartTV

The **SmartTV** represents a multimedia entertainment device that can be centrally controlled. It demonstrates how non-essential devices can be automatically managed, for example by turning off when not in use to conserve energy.

2.3.5 PlayStation

The **PlayStation** simulates a gaming console within the smart home. As a high-energy consumer device, it is managed through centralized control and

energy-saving modes to prevent unnecessary power usage.

2.3.6 WashingMachine

The **WashingMachine** represents a household appliance that operates in defined cycles. It can be scheduled to run during off-peak hours or when renewable energy is available, supporting sustainable energy consumption.

2.3.7 MotionSensor

The **MotionSensor** detects movement within a room and serves as a trigger for automation rules. It enables scenarios such as activating lights or security systems when motion is detected.

2.3.8 SecurityAlarm

The **SecurityAlarm** enhances home safety by responding to security-related events. It can be automatically activated based on time schedules or user roles.

2.3.9 DoorLock

The **DoorLock** simulates a smart locking mechanism that allows controlled access to the home. It integrates with the authentication and security modules to ensure that only authorized users can unlock or lock doors.

2.3.10 GarageDoor

The **GarageDoor** represents an automated entry system that can be opened or closed remotely. It supports automation rules such as automatic closing after a certain time or when the user leaves the home.

2.3.11 SmartDevice (Abstract Class)

The **SmartDevice** abstract class defines the shared structure and behavior of all smart devices. It includes common attributes such as device ID, name, and status, and declares abstract methods that must be implemented by all concrete devices. This abstraction enables polymorphic control of devices through the central controller.

Sustainability Contribution Several devices interact with the energy management module to reduce unnecessary power consumption. Through automation rules, scheduling, and centralized control, the system promotes responsible and sustainable energy usage without compromising user comfort.

3 Object-Oriented Design

Object-Oriented Design is the foundation of the Smart Home Automation Simulator. The system was intentionally modeled using object-oriented concepts to reflect real-world relationships between users, devices, rooms, and energy components. This approach ensures modularity, flexibility, and ease of extension while maintaining a clear and logical structure.

3.1 Class Diagram

The class diagram provides a high-level structural view of the system and illustrates the relationships between its main components. It highlights inheritance, aggregation, composition, and interface implementation used throughout the project.

At the core of the system lies the `SmartDevice` abstract class, which defines the common attributes and behaviors shared by all smart devices. Concrete device classes such as `Light`, `Thermostat`, `SmartTV`, `AirConditioner`, and `WashingMachine` inherit from this abstract class and provide their own implementations of device-specific behavior.

The `Home` class represents the global smart home environment and aggregates multiple `Room` objects. Each `Room` maintains a collection of smart devices, reflecting a real-world spatial organization. Additionally, the home may contain global devices such as door locks and security alarms that are not tied to a specific room.

User management is handled through an inheritance hierarchy starting with the abstract `User` class. Specialized roles such as `Admin`, `Parent`, `Child`, and `Guest` extend this class and override access-control logic, enabling role-based permissions within the system.

Automation and system coordination are managed by the `CentralController`, which implements the `AutomationController` interface. This controller interacts with the home model, user roles, and energy management components to apply automation rules and enforce system constraints.

Energy-related behavior is encapsulated within the `EnergyManager`, `BatteryStorage`, `SolarPanel`, and `EnergyMode` classes. These components collaborate to monitor energy production, storage, and consumption, allowing the system to dynamically adapt device behavior based on available resources.

3.2 OOP Principles Applied

The Smart Sustainable Home System was designed with a strong emphasis on Object-Oriented Programming (OOP) principles. These principles ensure that the system remains modular, extensible, reusable, and easy to maintain. The main OOP concepts applied in this project are abstraction, inheritance, polymorphism, encapsulation, and interfaces.

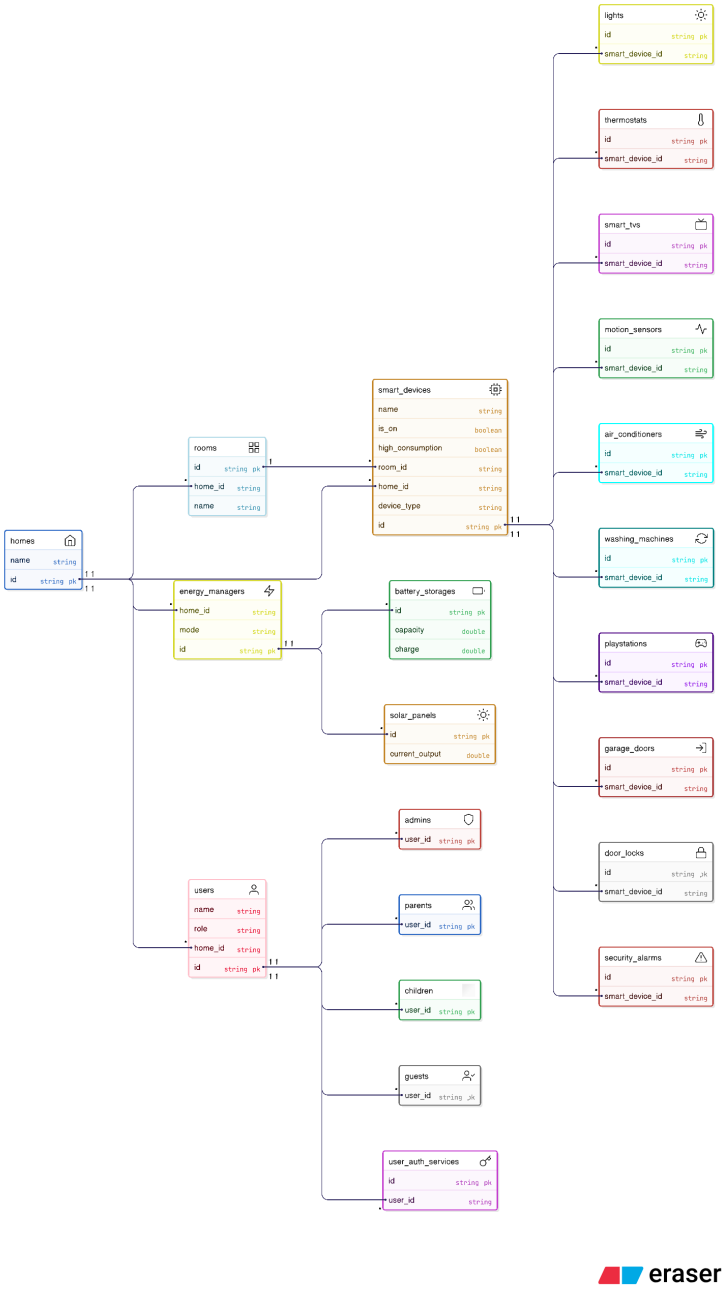

Figure 4: Class Diagram of the Smart Home Automation Simulator

3.2.1 Abstraction

Abstraction is used to hide complex implementation details and expose only essential behaviors to the rest of the system. In this project, abstraction is achieved through both abstract classes and interfaces.

The `SmartDevice` abstract class defines the common structure and behavior shared by all smart devices, such as device name, operational state, and power consumption logic. Concrete device classes (e.g., `Light`, `Thermostat`, `SmartTV`) inherit from this class and implement device-specific functionality.

Similarly, abstraction is applied through the `AutomationController` interface, which defines the contract for executing automation rules without specifying how these rules are implemented. This allows the system to rely on high-level behavior rather than concrete logic.

3.2.2 Inheritance

Inheritance is used to establish parent–child relationships between classes, reducing code duplication and promoting reuse. All smart devices inherit from the abstract `SmartDevice` class, allowing them to share common attributes and methods while extending or customizing behavior.

For example, devices such as `GarageDoor`, `AirConditioner`, and `WashingMachine` extend `SmartDevice` and override methods related to power consumption and operational behavior.

Inheritance is also applied in the security module. The abstract `User` class defines the general structure for system users, while concrete classes such as `Admin`, `Parent`, `Child`, and `Guest` extend it to implement role-specific access control policies.

3.2.3 Polymorphism

Polymorphism allows objects of different classes to be treated uniformly through a common interface or parent class. In this system, the central controller and the user interface manipulate devices using references of type `SmartDevice`, regardless of their concrete implementation.

At runtime, the appropriate method implementation is executed depending on the actual device type. For example, calling `getPowerConsumption()` or `turnOn()` on a `SmartDevice` reference triggers different behaviors for a `Light`, a `SmartTV`, or a `Thermostat`. This enables flexible device management without conditional logic based on device type.

Polymorphism is also applied through interfaces. The `CentralController` class implements the `AutomationController` interface, allowing automation rules to be executed through a common method signature while supporting different internal implementations.

3.2.4 Encapsulation

Encapsulation is used to protect internal data and control access to class attributes. All sensitive data, such as device state, energy levels, and user information, are kept private within their respective classes.

Access to these attributes is provided through public methods (getters and setters), ensuring that changes to internal data are controlled and validated. This prevents inconsistent system states and improves robustness.

For example, the internal status of a device cannot be modified directly; it can only be changed through controlled methods such as `turnOn()` and `turnOff()`, which enforce energy and access constraints.

3.2.5 Interfaces

Interfaces are used to define clear behavioral contracts within the system. The `AutomationController` interface specifies the `tick(LocalTime, User)` method, which represents the execution of automation rules based on time and user context.

By using an interface, the system allows different controllers or automation engines to be introduced in the future without modifying existing components. This supports extensibility and adheres to the principle of programming to an interface rather than an implementation.

3.2.6 Overall Design Impact

The combination of abstraction, inheritance, polymorphism, encapsulation, and interfaces results in a clean and scalable architecture. These principles enable the system to:

- Easily integrate new smart devices and user roles
- Enforce security and energy constraints consistently
- Reduce code duplication and improve readability
- Support future extensions such as advanced automation or analytics

By applying these object-oriented principles, the Smart Sustainable Home System achieves a realistic, maintainable, and sustainable simulation that reflects real-world smart home software design.

4 System Implementation

The system of the Smart Sustainable Home System focuses on translating real-world smart home concepts into a structured and modular software solution. The system was carefully designed to balance clarity, extensibility, and realism while strictly following Object-Oriented Programming principles.

4.1 System Objectives

The main
objectives of the system are:

- To model real-world smart home components in a logical and intuitive manner
- To ensure separation of concerns between user interface, business logic, and data models
- To promote reusability and extensibility through object-oriented design
- To integrate sustainability considerations into system behavior
- To enforce security and access control through user roles

These objectives guided all design decisions made throughout the development process.

4.2 Modular Package-Based Design

The system is organized into multiple packages, each responsible for a specific concern:

- **devices**: Contains the abstract `SmartDevice` class and all concrete device implementations.
- **energy**: Manages energy production, storage, and consumption through dedicated classes.
- **home**: Represents the physical structure of the house using `Home` and `Room` classes.
- **security**: Implements user roles and access control mechanisms.
- **controller**: Coordinates automation rules and system-wide behavior.
- **ui**: Handles user interaction through a JavaFX graphical interface.

This modular organization improves maintainability and allows each component to evolve independently without impacting the rest of the system.

4.3 Device-Centric Design

All smart devices are modeled as subclasses of the abstract `SmartDevice` class. This design choice ensures that all devices share a common structure while allowing specialized behavior.

Each device encapsulates:

- Its operational state (on/off or active/inactive)
- Its energy consumption behavior
- Its response to global energy modes

By interacting with devices through the `SmartDevice` abstraction, the system avoids tight coupling between components and supports polymorphic behavior.

4.4 User-Centered and Role-Based Design

The system design incorporates multiple user roles to reflect realistic household usage scenarios. The abstract `User` class defines a general user structure, while specific roles such as `Admin`, `Parent`, `Child`, and `Guest` extend this class.

Each role enforces access control rules that restrict or allow interaction with devices based on user permissions. This design ensures both security and usability while keeping authorization logic centralized and consistent.

4.5 Energy-Aware Design

Sustainability is integrated directly into the system design rather than treated as an external feature. The `EnergyManager` acts as a central decision-making component that influences device behavior based on battery levels and energy modes.

Devices automatically adapt their behavior according to the current energy mode, enabling system-wide energy optimization without requiring manual intervention from users.

4.6 Automation-Oriented Design

Automation is designed around the `AutomationController` interface, which defines how automation rules are executed. The `CentralController` implements this interface and applies automation logic based on time, user context, and system state.

This design allows automation logic to be extended or replaced in the future without affecting existing components, supporting long-term scalability.

4.7 Design Benefits

The chosen design approach provides several advantages:

- Clear separation of responsibilities
- Reduced code duplication through inheritance
- Flexible extension of devices and automation rules
- Improved system readability and maintainability
- Seamless integration of sustainability and security concerns

Overall, the design of the Smart Sustainable Home System ensures a coherent, scalable, and realistic simulation that aligns with both academic requirements and real-world smart home architectures.

5 Sustainability Features

Sustainability is a core objective of the Smart Sustainable Home System. The project integrates energy-aware decision-making and automation mechanisms to reduce unnecessary power consumption while maintaining user comfort. Rather than focusing solely on device control, the system promotes responsible energy usage through intelligent software design.

5.1 Energy-Aware Device Management

The system includes an energy management module composed of the `EnergyManager`, `BatteryStorage`, `SolarPanel`, and `EnergyMode` components. These elements work together to monitor energy production, storage, and consumption across the home.

Devices dynamically adapt their behavior based on the current energy mode (HIGH, MEDIUM, or ECO). In ECO mode, devices automatically reduce their power usage or deactivate non-essential features, contributing to lower overall energy consumption.

5.2 Renewable Energy Integration

Solar energy generation is simulated through the `SolarPanel` class, which continuously charges the battery. Stored energy is then used to power smart devices. This design reflects real-world renewable energy systems and highlights the importance of storing excess energy for later use.

The battery level directly influences system behavior. When the battery charge drops below defined thresholds, the system automatically restricts high-consumption devices and switches to more energy-efficient modes.

5.3 Automation for Energy Optimization

Automation rules further enhance sustainability by eliminating unnecessary device usage. For example, devices can be turned off automatically at specific times, or restricted based on user roles and system conditions. These automated decisions reduce human error and prevent energy waste.

Overall, the system demonstrates how software-driven automation can support sustainable living by optimizing energy usage without sacrificing functionality.

6 Testing and Validation

Testing and validation were conducted throughout the development process to ensure system correctness, reliability, and consistency with functional requirements. Given the simulation-based nature of the project, testing focused on behavioral validation rather than hardware integration.

6.1 Functional Testing

Each smart device was tested individually to verify correct behavior when turned on or off, proper response to energy mode changes, and accurate power consumption calculations. User role permissions were also tested to ensure that access control rules were correctly enforced for Admin, Parent, Child, and Guest users.

Automation rules were validated by simulating different scenarios, such as low battery levels, time-based events, and user interactions. These tests confirmed that the system responds correctly under varying conditions.

6.2 Integration Testing

Integration testing ensured smooth interaction between system components, including the UI, controllers, energy management module, and home model. Special attention was given to validating communication between the JavaFX interface and backend logic, ensuring that device states and energy information were updated in real time.

6.3 User Interface Validation

The graphical user interface was tested to verify usability and clarity. Device controls, energy indicators, and role-based visibility were validated to ensure that users receive accurate feedback and intuitive interaction. Error handling mechanisms, such as warnings for insufficient battery levels, were also tested to confirm appropriate system responses.

7 Limitations and Challenges

Despite achieving its objectives, the Smart Sustainable Home System faces several limitations and challenges, primarily due to the scope and simulation-based nature of the project.

7.1 Simulation Constraints

The system operates entirely as a software simulation and does not interact with real hardware devices or sensors. As a result, energy generation, consumption, and device behavior are approximations rather than exact real-world measurements.

7.2 Simplified Energy Model

While the energy management module demonstrates core sustainability concepts, the energy model remains simplified. Factors such as variable solar output, weather conditions, device aging, and real-time consumption patterns are not fully represented.

7.3 Limited Automation Complexity

Automation rules are currently rule-based and predefined. More advanced automation techniques, such as machine learning or adaptive behavior based on user habits, are not implemented due to project scope limitations.

7.4 Scalability Constraints

Although the system is designed to be extensible, large-scale deployments with hundreds of devices or distributed homes were not tested. Performance under such conditions may require further optimization.

7.5 Future Improvements

These limitations open opportunities for future enhancements, including real hardware integration, advanced energy analytics, intelligent learning-based automation, and cloud-based scalability. Addressing these challenges would further strengthen the system's realism and practical applicability.

8 Conclusion

The system we developed demonstrates how object-oriented programming (OOP) principles can be applied to design a modular, scalable, and intelligent solution for modern living. By integrating devices, energy management, user roles, and automation rules into a cohesive architecture, the project achieves several key outcomes:

- **Efficiency & Sustainability:** The dynamic energy management system ensures optimal use of solar power and battery storage, automatically adjusting consumption modes to balance comfort with sustainability.
- **Security & Access Control:** Role-based user authentication (Admin, Parent, Child, Guest) enforces clear boundaries, ensuring responsible device usage while maintaining household safety.
- **Automation & Intelligence:** The CentralController applies real-time rules illustrating how automation enhances daily routines.
- **Robust Design:** The modular folder structure (controller, devices, energy, home, security, UI, exceptions) reflects strong OOP practices, making the system easy to extend with new devices, rules, or user roles.
- **User Experience:** The JavaFX-based interface provides a clear, interactive dashboard for monitoring energy levels, controlling devices, and visualizing automation in action, supported by custom exception handling for reliability.

This project bridges *technology, sustainability, and human-centered design.* It not only showcases technical mastery of OOP concepts but also highlights how software can contribute to smarter, greener, and safer homes. With further extensions—such as machine learning for predictive energy usage or IoT integration—the system could evolve into a fully autonomous smart home platform, reinforcing the vision of sustainable living through intelligent automation.

9 References

References

[1] Sanatt-gitcode. *Home-automation-project-using-java-oops*. GitHub repository, 2023. Available at: https://github.com/sanatt-gitcode/home-automation-project-using-java-oops

[2] CodeVisionz. *Java Smart Home System: OOP with Encapsulation & Composition*. Tutorial, 2022. Available at: https://www.codevisionz.com

[3] IEEE Xplore. *Object-Oriented Modelling of Home Automation Management System*. IEEE Conference Paper, 2021.

[4] Rehman, U., Faria, P., Gomes, L., & Vale, Z. *Future of Energy Management Models in Smart Homes: A Systematic Literature Review*. Process Integration and Optimization for Sustainability, 2025.

[5] IEEE Xplore. *Design and Development of a Smart Home Energy Management System*. IEEE Transactions on Smart Grid, 2022.

[6] El-Azab, R. *Smart homes: potentials and challenges*. Clean Energy, Oxford Academic, 2021.

[7] IEEE Xplore. *Hybrid Approaches (ABAC and RBAC) Toward Secure Access Control in Smart Homes*. IEEE Access, 2020.

[8] MDPI. *An Attribute-Based Approach toward a Secured Smart-Home IoT Access Control and a Comparison with a Role-Based Approach*. Sensors Journal, 2021.

[9] Springer. *Review of Security and Privacy-Based IoT Smart Home Access Control*. SpringerLink, 2022.

[10] GeeksforGeeks. *Designing UI for Smart Home Devices*. Tutorial, 2023. Available at: https://www.geeksforgeeks.org

[11] CodingTechRoom. *Implementing a Smart Home Automation System with Java and AI*. Blog, 2024. Available at: https://codingtechroom.com