Problem Statement & Objectives

Problem Statement: The increasing adoption of IoT devices in smart homes has led to a need for a centralized and efficient way to monitor and control multiple devices. Current solutions often lack real-time updates, seamless device interaction, and a user-friendly interface. This project aims to develop an IoT-enabled smart home dashboard with real-time WebSockets communication and an MVC-based architecture for structured development and scalability.

Objectives:

1. Develop a smart home dashboard with a user-friendly interface for monitoring and controlling IoT devices.
2. Implement WebSockets for real-time data updates and device status synchronization.
3. Utilize MVC architecture to ensure modularity, maintainability, and scalability.
4. Provide role-based authentication and authorization for secure access control.

Use Case Diagram & Descriptions

Use Case Diagram: The system consists of the following actors:

- **Homeowner (User):** Controls and monitors IoT devices.
- **IoT Devices:** Communicate with the dashboard to send and receive commands.

Use Case Descriptions:

1. **User Login & Authentication:** The homeowner logs into the system using secure authentication.
2. **Device Monitoring:** The user can view real-time status updates for all connected smart home devices.
3. **Device Control:** The user can turn devices on/off, change settings, and schedule automation.
4. **WebSocket Communication:** Ensures instant status updates and bidirectional communication between the dashboard and IoT devices.

Functional & Non-Functional Requirements

Functional Requirements:

1. User authentication and role-based access control.
2. Real-time device monitoring and control using WebSockets.
3. Device scheduling and automation.
4. Secure communication between the frontend, backend, and IoT devices.
5. User-friendly UI with intuitive controls.
6. Logging and reporting for device activity tracking.

Non-Functional Requirements:

1. **Performance:** The system should support at least 100 concurrent users with low latency.
2. **Security:** End-to-end encryption and OAuth-based authentication.
3. **Scalability:** The architecture should allow easy integration of new IoT devices and features.
4. **Reliability:** Maintain system uptime of at least 99.5%.
5. **Usability:** The UI should be accessible to both tech-savvy and non-tech users.
6. **Maintainability:** The MVC pattern ensures code modularity and easy debugging.

Software Architecture

Architecture Overview: The system follows the **Model-View-Controller (MVC)** architecture to ensure modular design and separation of concerns. It integrates WebSockets for real-time communication.

Key Components:

1. **Frontend (View):**
    - Developed using HTML, CSS, JavaScript, and a modern frontend framework
    - Implements WebSockets to receive real-time updates.
    - Provides a responsive and intuitive UI for users.
2. **Backend (Controller & Model):**
    - Developed using **ASP.NET Core MVC**.
    - Handles authentication, authorization, and user management.
    - Manages IoT device state and executes commands.
    - Uses WebSockets for real-time bidirectional communication.
3. **Database (Model):**
    - SQL Server \for structured data storage.
    - Stores user data, device configurations, and logs.
4. **WebSockets Communication Layer:**
    - Ensures real-time updates and event-driven device control.
    - Handles bi-directional communication between frontend and backend.
5. **IoT Device Integration:**
    - IoT devices communicate with the backend using WebSockets.