Smart Home Automation and Monitoring System

Documentation

Table of Contents

[Introduction 3](#_Toc180579140)

[System Architecture 4](#_Toc180579141)

[Backend Setup (ASP.NET Core Web API) 5](#_Toc180579142)

[Frontend Setup (Angular) 6](#_Toc180579143)

[API Reference 7](#_Toc180579144)

[User Manuals 8](#_Toc180579145)

[Conclusion 9](#_Toc180579146)

# Introduction

**1.1 Project Overview**

The **Smart Home Automation and Monitoring System** is designed to provide users with the ability to control and monitor various smart devices within their home environment. These devices include lights, thermostats, cameras, and other IoT-enabled systems. Users can interact with the system via a web interface that supports real-time monitoring and remote control of devices. The system includes features like:

* Device control (on/off, dimming, etc.)
* Scheduling devices to operate at specific times
* Monitoring devices’ statuses and receiving alerts
* User management with roles and permissions (admin, regular user)

This project is built on the following technologies:

* **ASP.NET Core Web API**: Backend API providing endpoints for device management, user authentication, and more.
* **Angular**: Frontend framework used for building the user interface, including dashboards, control panels, and configuration screens.
* **SQL Server**: The database for storing user information, device data, and scheduling configurations.

**1.2 Key Features**

* **Real-Time Monitoring**: Users can view the status of all connected devices in real-time.
* **Device Scheduling**: Set start and stop times for devices to automate their operation.
* **User Management**: Admin users can manage other users, assign roles, and configure devices.
* **Mobile and Web Compatibility**: The system is accessible from both desktop and mobile web browsers.
* **Cloud-Based System**: Supports cloud integration for secure storage and scalability.

# System Architecture

## Overview

The system follows a three-layered architecture:

* **Frontend (Angular)**: The user-facing application where users interact with the system. The UI is designed for ease of use, offering a dashboard for controlling devices and monitoring their statuses.
* **Backend (ASP.NET Core Web API)**: The middle layer where business logic and API endpoints are handled. This layer processes incoming requests from the frontend and interacts with the database.
* **Database (SQL Server)**: The database layer stores user information, device settings, event logs, and more. Entity Framework Core is used for managing database interactions and migrations.

## Components

* **ASP.NET Core Web API**:

\* Handles requests and responses.

\* Implements business logic and data access.

* **Angular Application**:

\* Renders the user interface.

\* Interacts with the backend via HTTP requests.

* **SQL Server**:

\* Stores application data securely.

\* Supports efficient querying and data manipulation. **User Interface (Angular)**: The web application where users can log in, control devices, set schedules, and view real-time updates.

# Backend Setup (ASP.NET Core Web API)

## Prerequisites

* [.NET SDK](https://dotnet.microsoft.com/download) (version 5.0 or higher)
* [SQL Server](https://www.microsoft.com/en-us/sql-server/sql-server-downloads)
* IDE (Visual Studio or Visual Studio Code)

## Project Structure

The project is organized into several key folders:

* **Controllers**: Handles the HTTP requests for devices, users, and authentication.
* **Models**: Defines the structure of data entities like Device, User, etc.
* **Services**: Contains the core business logic, such as device management and scheduling.
* **Data**: Manages the database context and migrations, using Entity Framework Core.
* **Middleware**: Custom middleware for handling errors, logging, and security.

## Configuration

**Install Packages**: Use NuGet Package Manager to install required packages like \*Microsoft.EntityFrameworkCore.SqlServer

\*Microsoft.EntityFrameworkCore.Tools

\*Microsoft.EntityFrameworkCore

**appsettings.json**: Updated with Sqlserver, name database name and added connectionstrings

**Dependency Injection**: Configure services and the database context in Program.cs to set up dependency injection.

## Database Setup

**Create Database**: Use SQL Server Management Studio to create a new database named SmartHomeAPI2DB.

**Migrations**: Run the following commands to set up your database tables:

* Add-Migration InitialCreate
* Update-Database

# Frontend Setup (Angular)

## Prerequisites

\* [Node.js](https://nodejs.org/) (version 14.x or higher)

\* Angular CLI

\* Visual Studio Code

## Project Structure

The Angular application is structured into several modules and components:

* **src/app/components**: Contains components like DevicesComponent, DashboardComponent, login and register components, etc., which are responsible for rendering the device dashboard and user interface.
* **src/app/services**: Houses services like DeviceService and UserService, which handle HTTP requests to the backend.
* **src/app/models**: Defines TypeScript interfaces and models that represent Device, User, and other entities.

## Configuration

**Install Angular Material**:

\* ng add @angular/material

**Set up HTTP Client**: Ensure HttpClientModule is imported in your main application module:

import { HttpClientModule } from '@angular/common/http';

@NgModule({

imports: [

HttpClientModule,

// other modules

]

})

# API Reference

## Authentication

**Login**: POST /api/User/login: Registers a new user in the system

**Request Body**: { "username": "johndoe", "password": "secret" }

**Response**: 200 OK, returns a JWT token (if implemented).

**Register**: POST /api/User/register: Registers a new user in the system.

**Request Body**: { "username": "johndoe", "password": "secret", "isAdmin": "boolean" }

**Response**: 201 Created, returns the user details.

## Device Management

**GET /api/devices**: Retrieves the list of all devices.

* **Response**: 200 OK, returns a list of devices.

**POST /api/devices**: Adds a new smart device to the system.

* **Request Body**: { "name": "Living Room Light", "status": "off", ”isActive” }
* **Response**: 201 Created, returns the created device details.

## User Management

**Get Users**: GET /api/user

**Update User**: PUT /api/user/{id}

**Delete User**: DELETE /api/user/{id}

# User Manuals

## Admin User Guide

Provide detailed instructions for administrators on managing the system.

1. **Login**: Admin users can log in via the /admin endpoint using their credentials.
2. **Add Device**: Navigate to the “Devices” section, click “Add Device”, and fill in the necessary details like device name and type.
3. **Manage Users**: Admins can create, edit, or delete user accounts in the system.

## End User Guide

Provide instructions for regular users.

1. **Login**: Users can log in via the main login page using their credentials.
2. **Control Devices**: After login, users can control devices by switching them on/off or configuring their schedules.
3. **Monitor Devices**: Users can view the current status of their devices, such as power usage and operating state.

# Conclusion

The Smart Home Automation and Monitoring System offers a comprehensive solution for managing and controlling IoT devices within a home environment. Built using modern web technologies like ASP.NET Core, Angular, and SQL Server, it ensures scalability and real-time performance. Future iterations could incorporate support for voice assistants and external integrations with third-party smart home services like Amazon Alexa and Google Home.