House Automation App Documentation

1. Introduction

This is a single page application that is simulating house automation, that will allow remote clients (iPad browser web app for example) to monitor and control home appliances. It is built using the React framework of JavaScript. It includes HTTP-based API interactions that are simulated using dummy data. The UI is styled using SCSS modules. Node-Sass is used which allows .scss files to be natively compiled to css.

2. Folders

As with ReactJS, the app uses components to present data and containers for API calls. The folders contained in src directory are as follows:

Components: This folder contains the main UI and controls of the application and used in extending the app.
Containers: This folder contains components that react with Redux store, they represent controllers in an MVC model.
Hoc: This is a High-Order-Components and work as decorators

Store: This contains redux store reducers, actions types and action creators.
Styles: This represents the general styles
Utils: This contains API functions, helper functions, application contents and FontAwesomeIcons exports.

3. Models

Rooms

This contains all the rooms of the house. Data includes: name for room name, icon for Fontawesome icon and devicesCount to represent number of devices.

Devices

This contains the devices in a room. Data fields include: name, icon, switch to show whether the device is on or off and controls for various controls of a device

Controls

This contains the controls that are required. Fields include name, type and value. Controls include temperature control and mode control.

4. API Endpoints

The application uses axios library to perform HTTP API calls. API functions can be found in src/utils/api folder. The axios.js file initializes the axios instance used in the application.

- getRoomsApi(): GET Request: returns the rooms in the house
- getRoomDevicesApi(roomId): GET Request: returns the devices of a specified room with room id.
- toggleDeviceSwitchApi(deviceId): PATCH Request: changes the state of the device either on or off when passed the device id.
- updateDeviceControlValueApi(payload): PATCH Request changes the value of a certain control

API error requests are handled by the global Modal component in src/hoc/Layout/Layout.js file.

5. Extending Application

The application can be extended in three main categories: Controls Types, API Endpoints and Redux store. To add extra rooms, devices and controls, they are retrieved by calling endpoints to get data from their API.

How to add Device Control

- Create the component folder in src/components/Device/Controls folder
- Add the following files [control].js and [control].module.scss with [control] as name of needed control
- The component is passed prop onUpdateValue which must be called when the value is changed and it should pass the deviceId and the updated value.
- Add the device type constant in src/utils/deviceControls.types.js.
- Add the control into the ControlsSwitcher component at src/components/Device/ControlsSwitcher/ControlsSwitcher.js in the switch case to select the component when the type matches.

To add API Endpoints, add new endpoint calls in src/utils/api
base_url can be changed in src/config.js

To add new redux actions, reducers and action types use src/store/