Smart Home Mockup

Setup

Before beginning the project, setting up the project and the development environment is necessary. Thus, the following steps were taken:

- Installing the latest version of the Angular CLI
- Creating a Git repository
- Creating a log (this document) in which in to take notes regarding the development process

Part 1

The first part of this project consisted of recreating the two cards shown in the picture and implementing functionality for changing the color of the small red color indicator belonging to the first card. In continuation, these are the steps needed to accomplish these tasks:

1) Created new Angular project called *smart-home-angular* using the Angular CLI in the parent folder *smart-home*
2) In the newly generated *index.html* file, add links to Font Awesome icons and the Lato Google Font
3) Used Angular router to easily setup navigation for the following pages:
    - Home – basic home page for the application
    - Network - page where all the devices associated to the network are shown
    - Page Not Found – page used for handling incorrect URLs
4) Established the core structure of the application component tree, which contains the following components:
    - Home Component
    - Nav Component
    - Network Component – has nested components representing the network devices:
        - Aalborg Device Component – represents the first card in the picture
        - Generic Device Component – detailed in Part 2
        - Generic Value Component – also detailed in Part 2
        - Lights Device Component - represents the second card in the picture
    - Page Not Found Component
5) For recreating the mentioned cards, I used plain CSS and Angular Directives such as *ngIf* and *ngStyle*. When it comes to styling, I made use of both Flexbox and Grid displays along with Sub Grids to achieve the wanted look for the cards.
6) For changing the color of small color indicator, I used a random number to determine when to set the color to red or gray depending on its value.
7) For ensuring responsiveness I used the Flexbox display in the Network page which allows its components to reposition themselves depending on the screen size.

Part 2

In the second part of this project, the focus was placed on displaying the data provided by the IoT Demo REST API. Using the Angular CLI, it is possible to serve the application using a local server which updates itself automatically when it detects changes. This server is located by default at *http://localhost:4200/*, and thus, this caused a CORS issue with the API, which is located at a different host. In this case, disabling CORS on the server side was not an option, therefore I had to setup a proxy to be able to access data from the API.

When dealing with APIs, it is customary in Angular to create services that will be injected in the components that require the API data using the Dependency Injection pattern. I created a service called *SeluxitService*, which is responsible for making API calls to the respective endpoints for getting the necessary data.

An instance of the SeluxitService is injected in the Network Component. By implementing the Observer pattern, the component calls the service which returns an Observable of type network to which it subscribes to receive network data from the API. For representing the API data on the Frontend side, I used models which are just TypeScript classes with various attributes. *ngFor* is used to go through the device data associated with the Living room network, which is then passed to the Generic Device Component. This component takes a device as input and creates a card for the device as well as listing its values and states in a style consistent with the previously recreated cards. Passing data from parent components to child components is possible due to Angular data binding. The Network Component also subscribes to receive updates regarding report state events from the Web Socket. Whenever new data is available, it is passed down to the Generic Value Component using Input and data binding which is how updating the views of the application takes place.

To display more information regarding the device values, I created the Generic Value Component, which does the same job as the Generic Device Component, except that it's adapted to work with values. Displaying API data is currently done only using the Generic Value Component. This component takes as inputs both a device and its respective value to function properly. The reason why I require the device is for its *deviceID*, which is used for updating control state data.

Observations regarding the *IoT Demo REST API* data:

- Oftentimes there are inconsistencies in the way that the data is represented, some examples being the following:
    - For most values and states, the Report state is listed **before** the Control state if both exist, however for the *Heat level* value and *Desired Temperature* value, the opposite is true
    - For the *Desired Temperature* and *Smoke Temperature* values of the *Stove-2020* device there is no unit specified even though for the *Temperature* value of the *Indoor module* device there is the Celcius (I am assuming that Celsius is misspelled on purpose) unit specified. It is worth noting that all the mentioned values have the same value type *temperature*.

    - Inconsistent value types and inappropriate/pointless values
        - In the case of the *Status* value for the *Stove-2020* device – this value has the type “state” which is seems wrong given the way that the Wappsto UDM functions. I think that it would be more suitable for this value to be represented as a Report state for *the On/Off* value
        - Both the *Indoor module* device and the *Stove-2020* device have a value called *Alarm* which seems to be used in the same way in both cases. Thus, it seems strange that the *Alarm* value of the first device has the type “message” and the *Alarm* value of the second device has the type “state”
- The Web Socket is designed to send updates concerning Report state events however in the case of the *On/Off* value which has only a Control state this means that the view does not get updated the Control state is changed because this change is not detected by the Web Socket