Test #1 | RDM Device List

Development Requirements

- Node.js

Project Files

```
DeviceList
 ├ src
 │ ├ res // Ressource files (css/fonts/images)
 │ ├ DynamicList.ts // Dynamic list implementation
 │ ├ index.html
 │ ├ main.ts       // App entry point
 │ ├ RDM_Device.ts  // RDM Device type definition
 │ └ Server.ts      // Backend server interface (test data generator)
 ├ package-lock.json
 ├ package.json
 ├ tsconfig.json
 └ webpack.config.js
```

Project Setup

1. Download and extract the project template (`NA_DeviceList_Test.zip`)
2. Start a terminal instance in the `./DeviceList` directory and run `npm update` to download all required Node.js dependencies

Build and Run

1. Start a terminal instance in the `./DeviceList` directory
2. Run `npm run start`

> Webpack is used to bundle this TypeScript project into JavaScript
> Running the `npm run start` command will build the project and open a live development server instance in your default browser. Everything is configured to rebuild the project and refresh the browser instance automatically on any file update.

Project Template

> This project template does not use any frameworks.
> You are allowed to add any HTML/CSS/JS/TS frameworks/modules if needed, but **TypeScript must remain the primary script language**.
> You are also allowed to modify any existing files/classes/interfaces/functions to better suit your style of doing things. (Any except `Server.ts` and `RDM_Device.ts`)

Terminology

RDM	**Remote Device Management** - a communication protocol used in stage lighting that allows discovery and remote management of lighting fixtures and other devices.
RDM Device	A device that has been discovered in an RDM network. An RDM Device contains multiple data fields, such as, is-online state, label, UID, manufacturer and others.

"Backend" Interface

`Server.ts` contains a class that simulates discovery of new devices and updates of their parameters.

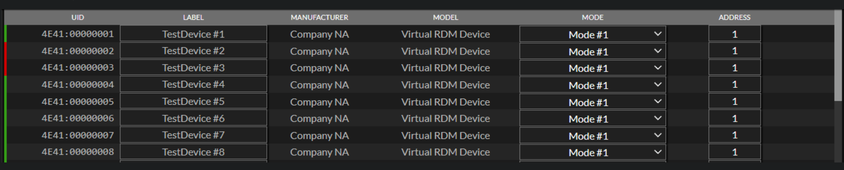
```ts
// [main.ts]
g_Server = new Server({
    device_added_callback: (device_data: RDM_Device) => {
        // Called when a new RDM Device has been discovered.
        // Create an RDM Device entry in the RDM Device List with the values in device_data.
        console.log("Add Device", device_data)
    },
    device_updated_callback: (device_data: RDM_Device) => {
        // Called when an RDM Device parameter change is detected.
        // Update existing associated RDM Device entry in the RDM Device List with the values in device_data.
        console.log("Update Device", device_data)
    }
})
```

The `RDM_Device` data structure is defined in `./RDM_Device.ts` and is used to keep track of discovered devices and their current parameter values

```ts
// [RDM_Device.ts]
export interface RDM_Device {
    uid_integer: BigInt   // Unique ID integer value
    uid: string           // Unique ID hexadecimal formatted string value (ABCD:12345678)
    is_online: boolean,   // true/false
    label: string         // Device Label
    manufacturer: string  // Device Manufacturer
    model: string         // Device Model Name
    mode_index: number    // Device Mode Index
    address: number       // Device Address
}
```

RDM Device Entry

`RDM Device` entry elements should look like this:



UID	LABEL	MANUFACTURER	MODEL	MODE	ADDRESS
4E41:00000001	TestDevice #1	Company NA	Virtual RDM Device	Mode #1	1
4E41:00000002	TestDevice #2	Company NA	Virtual RDM Device	Mode #1	1
4E41:00000003	TestDevice #3	Company NA	Virtual RDM Device	Mode #1	1
4E41:00000004	TestDevice #4	Company NA	Virtual RDM Device	Mode #1	1
4E41:00000005	TestDevice #5	Company NA	Virtual RDM Device	Mode #1	1
4E41:00000006	TestDevice #6	Company NA	Virtual RDM Device	Mode #1	1
4E41:00000007	TestDevice #7	Company NA	Virtual RDM Device	Mode #1	1
4E41:00000008	TestDevice #8	Company NA	Virtual RDM Device	Mode #1	1

Each row represents an `RDM Device` and contains these columns:

Column	RDM_Device Field	Description
Blank; Red/Green	`.is_online`	Red/green online status indicator element
UID	`.uid`	String
LABEL	`.label`	Text input element
MANUFACTURER	`.manufacturer`	String
MODEL	`.model`	String
MODE	`.mode_index`	Select element (containing `.mode_count` number of options)
ADDRESS	`.address`	Text input that accepts integers `1` to `512`

> Editing any of the text/select/number inputs should just log the `RDM Device` UID and modified value to console

RDM Device List

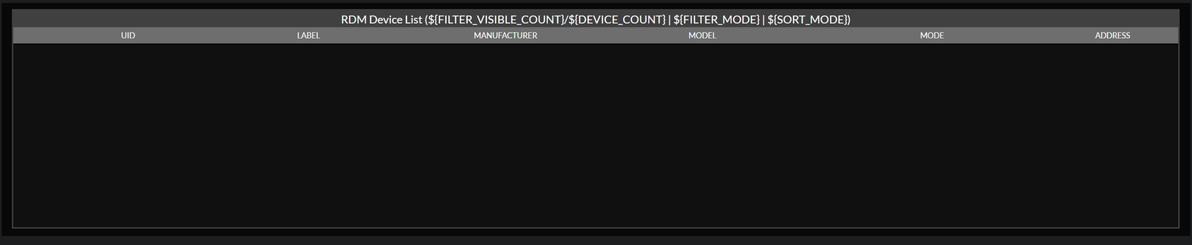
- The device list should be able to handle resizing of the browser window
- The table header should remain on top while scrolling

Test Assignment

- Write the required code for displaying an `RDM Device` entry (*see "RDM Device Entry" section for more information*).
- Create a scrollable list that is capable of displaying an arbitrary amount of `RDM Device` entries. The number of devices can potentially be very large, so keep performance in mind.
- Update the RDM Device List table title to show list device count, filtered device count, filter setting and sort mode.
- Implement "Sort By UID" test function for sorting list devices by `.uid_integer` value (in ascending order)
- Implement "Sort By Address" test function for sorting list devices by `.address` value (in ascending order)
- Implement "Sort By Manufacturer" test function for sorting list devices by `.manufacturer` value (in alphabetical order)
- Implement "Filter: None" test function to show all devices
- Implement "Filter: NA" test function to show only `.manufacturer == "Company NA"` devices
- Implement "Filter: TMB" test function to show only `.manufacturer == "TMB"` devices

> If multiple devices have the same sort condition value (for example, A.mode == 1 and B.mode == 1), then the sort order between them should be `.uid_integer` (in ascending order)
> Template implementation for button events is defined in `main.ts`

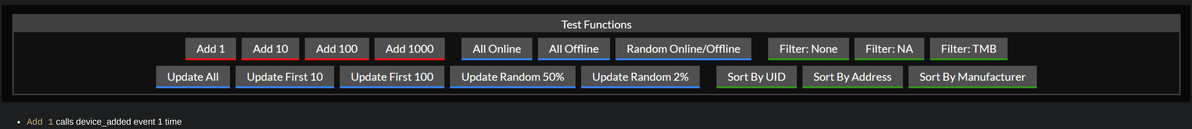
`index.html` and `res/index.css` contains a basic layout and table header:



RDM Device List (${FILTER_VISIBLE_COUNT}/${DEVICE_COUNT} | ${FILTER_MODE} | ${SORT_MODE})

UID	LABEL	MANUFACTURER	MODEL	MODE	ADDRESS

And test buttons for triggering server device add/update events and switching sort modes:



Test Functions

Add 1 | Add 10 | Add 100 | Add 1000 | All Online | All Offline | Random Online/Offline | Filter: None | Filter: NA | Filter: TMB

Update All | Update First 10 | Update First 100 | Update Random 50% | Update Random 2% | Sort By UID | Sort By Address | Sort By Manufacturer

- `Add 1` calls device_added event 1 time
- `Add 10` calls device_added event 10 times
- `Add 100` calls device_added event 100 times
- `Add 1000` calls device_added event 1000 times
- `All Online` calls device_updated event for all devices with `.is_online` == true
- `All Online` calls device_updated event for all devices with `.is_online` == false
- `Random Online/Offline` calls device_updated event for all devices with `.is_online` being a random true/false value
- `Update All` calls device_updated event for all devices with random device parameter values
- `Update First 10` calls device_updated event for first 10 devices with random device parameter values
- `Update First 100` calls device_updated event for first 100 devices with random device parameter values
- `Update Random 50%` calls device_updated event for 50% of random devices with random device parameter values
- `Update Random 2%` calls device_updated event for 2% random devices with random device parameter values
- `Filter: None` Set list filter (custom implementation required)
- `Filter: NA` Set list filter (custom implementation required)
- `Filter: TMB` Set list filter (custom implementation required)
- `Sort By UID` Set list sort mode (custom implementation required)
- `Sort By Address` Set list sort mode (custom implementation required)
- `Sort By Manufacturer` Set list sort mode (custom implementation required)

> All files except `Server.ts` and `RDM_Device.ts` can be modified to complete this assignment - you can add frameworks, modules, webpack plugins and anything else you might need.
> You can add code to `Server.ts` if you require a function like `GetDeviceByUID`, but do not modify any existing `Server` class functionality.

> IE support is not required