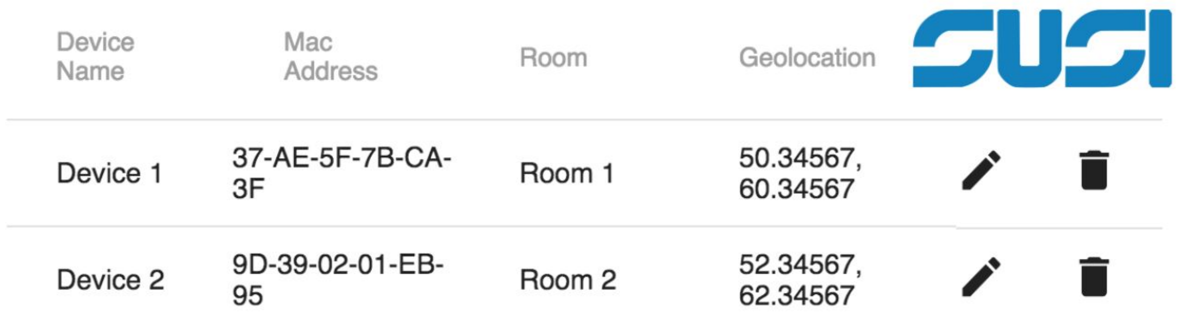

Device Name	Mac Address	Room	Geolocation		
Device 1	37-AE-5F-7B-CA-3F	Room 1	50.34567, 60.34567	✎	🗑
Device 2	9D-39-02-01-EB-95	Room 2	52.34567, 62.34567	✎	🗑

Implementing Table View in Devices Tab in Settings

We can connect to the SUSI.AI Smart Speaker using our mobile apps (Android or iOS). But there needs to be something implemented which can tell you what all devices are linked to your account. This is in consistency with the way how Google Home devices and Amazon Alexa devices have this feature implemented in their respective apps, which allow you to see the list of devices connected to your account. This blog post explains how this feature has been implemented on the SUSI.AI Web Client.

Fetching data of the connected devices from the server

The information of the devices connected to an account is stored in the Accounting object of that user. This is a part of a sample Accounting object of a user who has 2 devices linked to his/her account. This is the data that we wish to fetch. This data is accessible at the /aaa/listUserSettings.json endpoint.

```
{
  "devices": {
    "37-AE-5F-7B-CA-3F": {
      "name": "Device 1",
      "room": "Room 1",
      "geolocation": {
        "latitude": "50.34567",
        "longitude": "60.34567"
      }
    },
    "9D-39-02-01-EB-95": {
      "name": "Device 2",
      "room": "Room 2",
      "geolocation": {
        "latitude": "52.34567",
        "longitude": "62.34567"
      }
    }
  }
}
```

In the Settings.react.js file, we make an AJAX call immediately after the component is mounted on the DOM. This AJAX call is made to the /aaa/listUserSettings.json endpoint. The received response of the AJAX call is then used and traversed to store the information of each connected device in a format that would be more suitable to use as a prop for the table.

```
apiCall = () => {
    $.ajax({
      url: BASE_URL + '/aaa/listUserSettings.json?' + 'access_token=' +
  cookies.get('loggedIn');,
      type: 'GET',
      dataType: 'jsonp',
      crossDomain: true,
      timeout: 3000,
      async: false,
      success: function(response) {
        let obj = [];
         // Extract information from the response and store them in obj object
        obj.push(myObj);
        this.setState({
          dataFetched: true,
          obj: obj,
        });
      }
      }.bind(this),
  };
```

This is how the extraction of keys takes place inside the apiCall() function. We first extract the keys of the ‘devices’ JSONObject inside the response. The keys of this JSONObject are the Mac Addresses of the individual devices. Then we traverse inside the JSONObject corresponding to each Mac Address and store the name of the device, room and also the geolocation information of the device in separate variables, and then finally push all this information inside an object, namely ‘myObj’. This JSONObject is then pushed to a JSONArray, namely ‘obj’. Then a setState() function is called which sets the value of ‘obj’ to the updated ‘obj’ variable.

```
let keys = Object.keys(response.devices);
keys.forEach(i => {
    let myObj = {
        macid: i,
        devicename: response.devices[i].name,
        room: response.devices[i].room,
        latitude: response.devices[i].geolocation.latitude,
        longitude: response.devices[i].geolocation.longitude,
    };
}
```

This way we fetch the information of devices and store them in a variable named ‘obj’. This variable will now serve as the data for the table which we want to create.

Creating table from this data

The data is then passed on to the Table component as a prop in the Settings.react.js file.

```
<TableComplex
    // Other props
    tableData={this.state.obj}
/>
```

The other props passed to the Table component are the functions for handling the editing and deleting of rows, and also for handling the changes in the textfield when the edit mode is on. These props are as follows:

```
<TableComplex
    startEditing={this.startEditing}
    stopEditing={this.stopEditing}
    handleChange={this.handleChange}
    handleRemove={this.handleRemove}
/>
```

The 4 important functions which are passed as props to the Table component are:

1. **startEditing() function:**
   When the edit icon is clicked for any row, then the edit mode should be enabled for that row. This also changes the edit icon to a check icon. The columns corresponding to the room and the name of the device should turn into text fields to enable the user to edit them. The implementation of this function is as follows:

```
startEditing = i => {
    this.setState({ editIdx: i });
};
```

‘editIdx’ is a variable which contains the row index for which the edit mode is currently on. This information is passed to the Table component which then handles the editing of the row.

2. **stopEditing() function:**
   When the check icon is clicked for any row, then the edit mode should be disabled for that row and the updated data should be stored on the server. The columns corresponding to the room and the name of the device should turn back into label texts. The implementation of this function is as follows:

```
stopEditing = i => {
  let data = this.state.obj;
  this.setState({ editIdx: -1 });
  // AJAX call to server to store the updated data
  $.ajax({
    url:
      BASE_URL + '/aaa/addNewDevice.json?macid=' + macid + '&name=' + devicename
+ '&room=' + room + '&latitude=' + latitude + '&longitude=' + longitude +
'&access_token=' + cookies.get('loggedIn'),
    dataType: 'jsonp',
    crossDomain: true,
    timeout: 3000,
    async: false,
    success: function(response) {
      console.log(response);
    },
  });
};
```

The value of ‘editIdx’ is also set to -1 as the editing mode is now off for all rows.

3. **handleChange() function:**
   When the edit mode is on for any row, if we change the value of any text field, then that updated value is stored so that we can edit it without the value getting resetted to the initial value of the field.

```
handleChange = (e, name, i) => {
  const value = e.target.value;
  let data = this.state.obj;
  this.setState({
    obj: data.map((row, j) => (j === i ? { ...row, [name]: value } : row)),
  });
};
```

4. **handleRemove() function:**
   When the delete icon is clicked for any row, then that row should get deleted. This change should be reflected to us on the site, as well as on the server. Hence, The implementation of this function is as follows:

```
handleRemove = i => {
  let data = this.state.obj;
  let macid = data[i].macid;

  this.setState({
    obj: data.filter((row, j) => j !== i),
  });
  // AJAX call to server to delete the info of device with specified Mac Address
  $.ajax({
    url:
      BASE_URL + '/aaa/removeUserDevices.json?macid=' + macid + '&access_token='
+ cookies.get('loggedIn'),
    dataType: 'jsonp',
    crossDomain: true,
    timeout: 3000,
    async: false,
    success: function(response) {
      console.log(response);
    },
  });
};
```

This is how the Table View has been implemented in the Devices tab in Settings on SUSI.AI Web Client.

Resources

- Material-UI Table Component - https://material-ui.com/demos/tables/
- Sample example on editing and deleting rows - https://github.com/benawad/basic-react-form/tree/6_edit_delete_rows
- Icons - https://material.io/tools/icons/

Tags:

FOSSASIA, SUSI.AI, Tutorial, GSoC, Devices, Table, AJAX, Settings