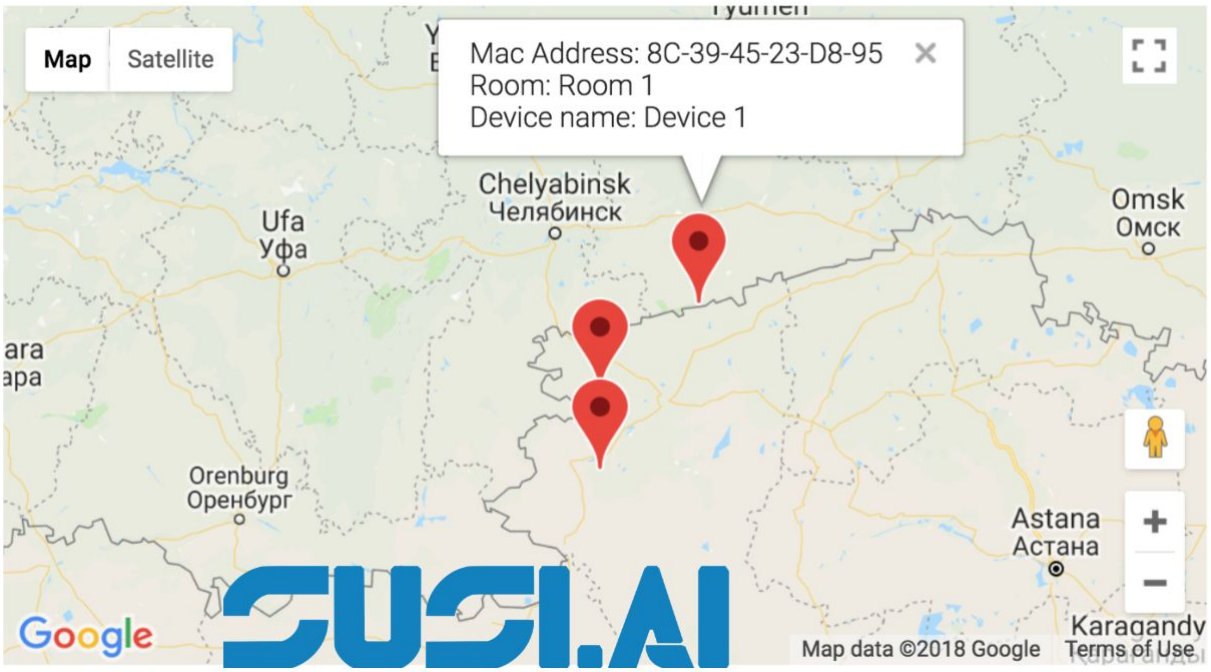

Implementing Map View in Devices Tab in Settings

The Table View implemented in the Devices tab in settings on SUSI.AI Web Client has a column “geolocation” which displays the latitudinal and longitudinal coordinates of the device. These coordinates needed to be displayed on a map. Hence, we needed a Map View apart from the Table View, dedicated to displaying the devices pinpointed on a map. This blog post explains how this feature has been implemented on the SUSI.AI Web Client.

Modifying the fetched data of devices to suitable format

We already have the fetched data of devices which is being used for the Table View. We need to extract the geolocation data and store it in a different suitable format to be able to use it for the Map View. The required format is as follows:

```
[
   {
      "location":{
         "lat": latitude1,
         "lng": longitude2
      }
   },
   {
      "location":{
```

```
        "lat": latitude1,
        "lng": longitude2
      }
   }
]
```

To modify the fetched data of devices to this format, we modify the apiCall() function to facilitate extraction of the geolocation info of each device and store them in an object, namely 'mapObj'. Also, we needed variables to store the latitude and longitude to use as the center for the map. 'centerLat' and 'centerLng' variables store the average of all the latitudes and longitudes respectively. The following code was added to the apiCall() function to facilitate all the above requirements:

```
let mapObj = [];
let locationData = {
  lat: parseFloat(response.devices[i].geolocation.latitude),
  lng: parseFloat(response.devices[i].geolocation.longitude),
};
centerLat += parseFloat(response.devices[i].geolocation.latitude);
centerLng += parseFloat(response.devices[i].geolocation.longitude);
let location = {
  location: locationData,
};
mapObj.push(location);
centerLat = centerLat / mapObj.length;
centerLng = centerLng / mapObj.length;
if (mapObj.length) {
  this.setState({
    mapObj: mapObj,
    centerLat: centerLat,
    centerLng: centerLng,
  });
}
```

The following code was added in the return function of Settings.react.js file to use the Map component. All the modified data is passed as props to this component.

```
<MapContainer
  google={this.props.google}
  mapData={this.state.mapObj}
  centerLat={this.state.centerLat}
  centerLng={this.state.centerLng}
  devicenames={this.state.devicenames}
  rooms={this.state.rooms}
  macids={this.state.macids}
/>
```

The implementation of the MapContainer component is as follows:

```
componentDidUpdate() {
   this.loadMap();
 }

 loadMap() {
   if (this.props && this.props.google) {
     const {google} = this.props;
     const maps = google.maps;
     const mapRef = this.refs.map;
     const node = ReactDOM.findDOMNode(mapRef);

     const mapConfig = Object.assign({},
       {
         center: { lat: this.props.centerLat, lng: this.props.centerLng },
         zoom: 2,
       }
     )
     this.map = new maps.Map(node, mapConfig);
   }
 }
```

Let us go over the code of MapContainer component step by step.

1. Firstly, the componentDidUpdate() function calls the loadMap function to load the google map.

```
componentDidUpdate() {
  this.loadMap();
}
```

2. In the loadMap() function, we first check whether props have been passed to the MapContainer component. This is done by enclosing all contents of loadMap function inside an if statement as follows:

```
if (this.props && this.props.google) {
  // All code of loadMap() function
}
```

3. Then we set the prop value to google, and maps to google maps props. This is done as follows:

```
const {google} = this.props;
const maps = google.maps;
```

4. Then we look for HTML div ref ‘map’ in the React DOM and name it ‘node’. This is done as follows:

```
const mapRef = this.refs.map;
const node = ReactDOM.findDOMNode(mapRef);
```

5. Then we set the center and the default zoom level of the map using the props we provided to the MapContainer component.

```
{
  center: { lat: this.props.centerLat, lng: this.props.centerLng },
  zoom: 2,
}
```

6. Then we create a new Google map on the specified node (ref='map') with the specified configuration set above. This is done as follows:

```
this.map = new maps.Map(node, mapConfig);
```

7. In the render function of the MapContainer component, we return a div with a ref ‘map’ as follows:

```
render() {
   return (
     <div ref="map" style={style}>
       loading map...
     </div>
   );
 }
```

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

Resources

- ‘google-maps-react’ library - https://www.npmjs.com/package/google-maps-react
- JavaScript Objects - https://www.w3schools.com/js/js_object_definition.asp
- AJAX - https://www.w3schools.com/xml/ajax_intro.asp

Tags:

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