**/\* Socket io Dashboard Real time widget data \*/**

**Subscription Method :-**

widgetTopicSubscription(widgetConfig) {

// Unsubscribe widget

this.unsubscribeWidgetTopic(widgetConfig.id);

// Subscribe widget with realtime

const topicSubscriptionUrl = this.customDataTopicName + '/' + widgetConfig.ids + '/' + widgetConfig.type + '/' + widgetConfig.span + '/' + widgetConfig.operation + '/' + widgetConfig.rollup + '/' + widgetConfig.source;

this.socketService.sendMessage({

topic: this.customDataTopicName,

ids: widgetConfig.ids,

type: widgetConfig.type,

span: widgetConfig.span,

operation: widgetConfig.operation,

rollup: widgetConfig.rollup,

source: widgetConfig.source

});

const topicSubInfo: WidgetTopicInfo = {

topic: topicSubscriptionUrl,

sub: null

}

topicSubInfo.sub = this.socketService.getMessage(topicSubscriptionUrl).subscribe((widgetData) => {

try {

// Info: Actual Data will get, don't need to call chartData method

const index = this.customDashboardService.allWidgetData.findIndex(w => w.id == widgetConfig.id);

if (index > -1) {

const widget = this.customDashboardService.allWidgetData[index];

const widgetOptions = widget.selectedOptions;

const chartType = widgetOptions.chartType;

let selectedChartData = this.getDefaultSelectedChartData(chartType, widgetOptions);

if (

chartType !== DashboardChartType.Digital &&

chartType !== DashboardChartType.Circular &&

chartType !== DashboardChartType.SemiCircular

) {

const chartData = this.createChartsData(widgetOptions, selectedChartData, widgetData);

selectedChartData = chartData.selectedChartData;

} else {

selectedChartData.chartLabels = [];

selectedChartData.chartData = [];

selectedChartData.legendNames = [];

if (widgetData && widgetData.length > 0) {

selectedChartData.chartData.push(

widgetData[0][this.getTotalizerParams(chartType, widgetOptions)]

);

}

}

this.customDashboardService.editWidgetEvent.next({

widgetData: { id: widget.\_id, name: widget.name },

widgetOptions: { ...widgetOptions, selectedChartData: selectedChartData },

index: index,

selectedChartData: selectedChartData,

chartType: chartType,

datapointIds: widgetOptions.datapointIds,

selectedNodesPreview: widgetOptions.selectedNodesPreview,

type: widgetOptions.type,

});

}

} catch (err) {

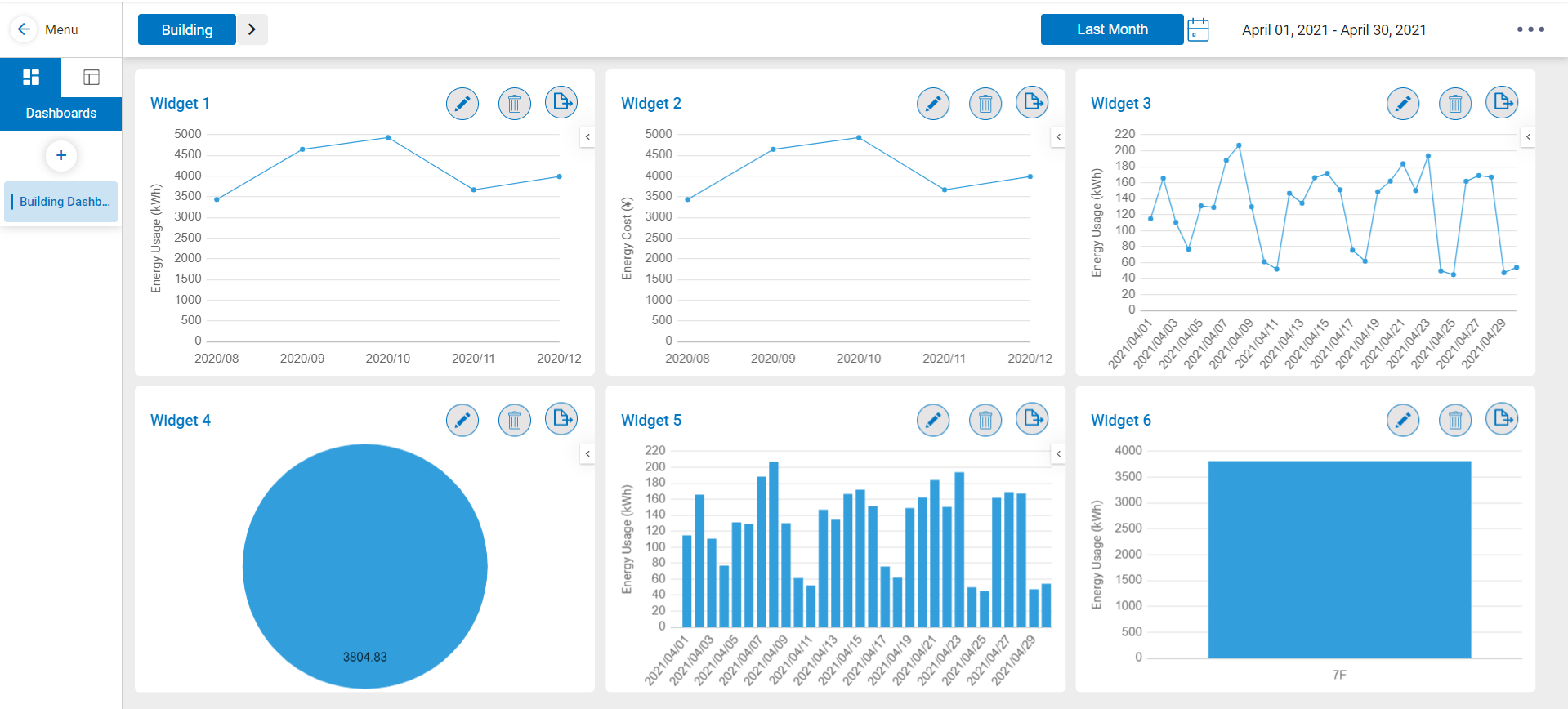
console.log(JSON.stringify(err));

}

});

this.widgetRealTimeDataSubs[widgetConfig.id] = topicSubInfo;

}



**Unsubscription Method :-**

unsubscribeWidgetTopic(widgetId) {

try {

const widgetTopicSubInfo: WidgetTopicInfo = this.widgetRealTimeDataSubs[widgetId];

if (widgetTopicSubInfo) {

this.socketService.sendMessage(

widgetTopicSubInfo.topic,

'unsubscribe'

);

if(widgetTopicSubInfo.sub) widgetTopicSubInfo.sub.unsubscribe();

}

} catch(error) {

}

delete this.widgetRealTimeDataSubs[widgetId];

}

**/\* Socket.service.ts file \*/**

import { Injectable } from '@angular/core';

import { Socket } from 'ngx-socket-io';

import { map } from 'rxjs/operators';

@Injectable({

providedIn: 'root'

})

export class SocketService {

constructor(private socket: Socket) { }

sendMessage(data, action = 'subscribe'){

this.socket.emit(action, data);

}

getMessage(topic) {

return this.socket

.fromEvent(topic)

.pipe(map((data: any) => data));

}

}