

Dokumentation zum Dürr HMI Framework

Inhaltsverzeichnis

1	Serverstruktur	4
1.1	Ordnerstruktur	4
2	Allgemeiner Seitenaufbau	6
2.1	Daten per json-Datei	7
2.2	Javascript/jQuery-Funktionen für den Seitenaufbau	8
2.3	SASS/CSS	14
3	Header Funktionen	15
3.1	Kunden-Logo / Dürr-Logo	15
3.2	Log-In	15
3.3	Konfiguration	17
3.3.1	Server	18
3.3.2	Framework	20
3.3.3	App	36
3.3.4	User	39
3.4	Hilfe	41
3.5	Sprachauswahl	42
3.6	AlarmOverlay	43
4	Main-Menü Funktionen	47
5	Content-Bereich Funktionen	48
5.1	Quick Alarm-Buttons	51
5.2	BreadCrumb-Navigation	52
5.3	OpenDiagnosisWindow	53
5.3.1	Das Informationsfenster	55
5.3.2	Fensterduplikate/ Vergleichsfenster	58
5.3.3	Scrollbars	59
5.3.4	Private-Tabs	64
5.3.5	Weitere Funktionen folgen	70
6	Funktionen	70
6.1	getTimeCodes(parent)	70
6.2	GetServerList()	71
6.3	NewServer()	71
6.4	newServerBody()	71
6.5	saveServerName(dataFile)	71
6.6	saveServer(dataFile, self)	71
6.7	deleteServerName(dataFile)	72
6.8	editServer(dataFile, editserver, parent)	72
6.9	showAdminpage(dataFile, parent)	72
6.10	editCentralCasServer(parent)	72
6.11	getLanguageList()	72
6.12	saveLanguageList()	72
6.13	saveLogo()	73
6.14	saveSettings()	73
6.15	saveOptions()	73
6.16	saveUserLanguage()	73
6.17	getprivateTabList()	73
6.18	getTrendingSetList()	73

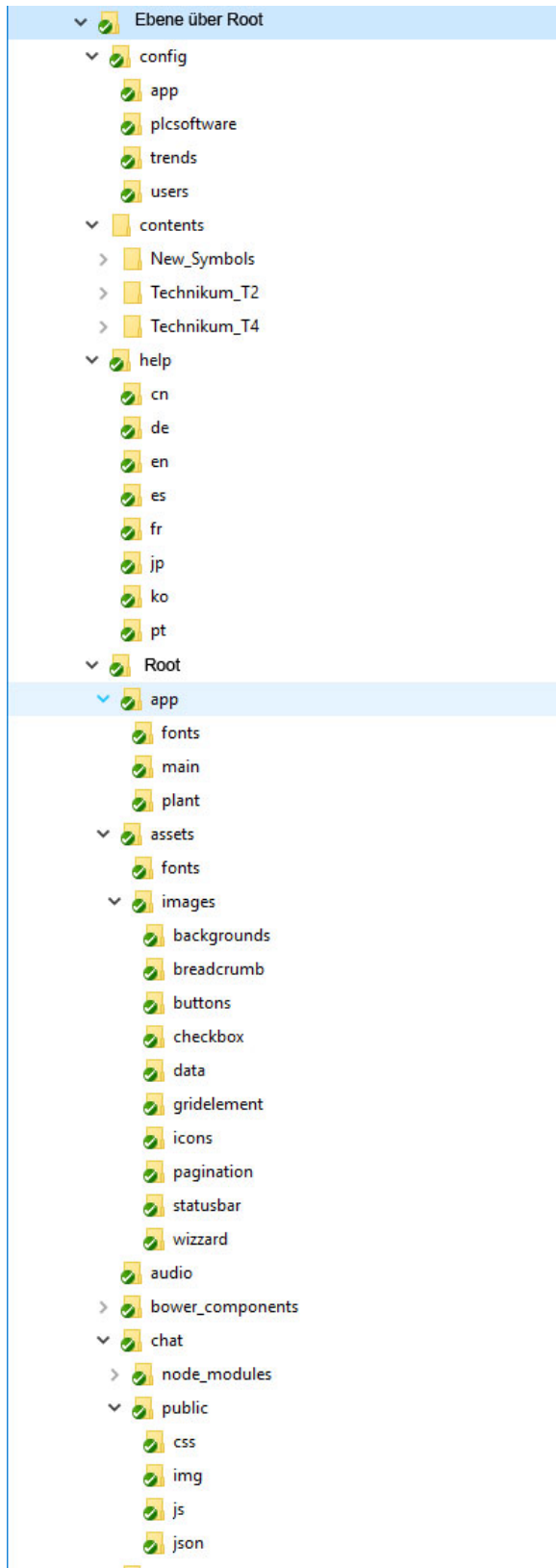
6.19	<u>saveTrendingSetFavorites()</u>	74
6.20	<u>setPlantFavorites()</u>	74
6.21	<u>savePlantFavorites()</u>	74
6.22	<u>buildLanguageDropDown()</u>	75
6.23	<u>getFolder(path)</u>	75
6.24	<u>buildAppList(parent)</u>	75
6.25	<u>buildClassList(path, id, selector)</u>	75
6.26	<u>buildButtonList(buttonList, parent, addClass, sidebuttonNum)</u>	75
6.27	<u>newApp(editapp)</u>	75
6.28	<u>addSidebutton(addNew, selector, datanum)</u>	75
6.29	<u>addPlant(addNew, sidebuttonNum, selector, datanum)</u>	76
6.30	<u>addAlarmbutton(addNew, sidebuttonNum, selector, datanum)</u>	76
6.31	<u>addHome(addNew, datanum)</u>	76
6.32	<u>getCancel(e, server)</u>	76
6.33	<u>hideElement(e)</u>	76
6.34	<u>buildInputGroup(valueText, textLabel, textName, setFloat)</u>	76
6.35	<u>editApp(parent, dataFile, duplicateFile)</u>	76
6.36	<u>editSidebutton(parent, dataFile, dataNum)</u>	77
6.37	<u>editPlant(parent, dataSidebuttonNum, dataFile, dataNum)</u>	77
6.38	<u>editAlarmbutton(parent, dataFile, dataNum, dataSidebuttonNum)</u>	77
6.39	<u>editHome(parent, dataFile, dataNum)</u>	77
6.40	<u>saveApp(parent)</u>	78
6.41	<u>activateConfig()</u>	78
6.42	<u>onOpenDiagnosisWindow(msg)</u>	78
6.43	<u>getDialog(mode, element, num)</u>	78
6.44	<u>getDeleteDialog(filename)</u>	78
6.45	<u>scrollfunc(e)</u>	78
6.46	<u>scrollUpDown(updown, myMainPage, myTarget)</u>	79
6.47	<u>getTransparenz(contentElement)</u>	79
6.48	<u>setScrollbar(contentElement)</u>	79
6.49	<u>scrollLeftRight(direction, nonDirection, nextVisibleElement, nextElement, newPosition, self)</u>	79
6.50	<u>getId()</u>	79
6.51	<u>clearMyInterval(self)</u>	79
6.52	<u>getAnimationData(name, tile)</u>	79
6.53	<u>uiMaker(myClass, myHtml, told, myID)</u>	79
6.54	<u>scrolltabs(self)</u>	80
6.55	<u>onMessage(msg, anotherWin)</u>	80
6.56	<u>showScroll(pageElement)</u>	80
6.57	<u>animateOff(layer, toHide, activeParent, notclose)</u>	80
6.58	<u>getHelpObject(cCode)</u>	80
6.59	<u>buildHelpLinks(helpResults)</u>	80
6.60	<u>closeOther(me)</u>	80
6.61	<u>buildHeader(pageld)</u>	81
6.62	<u>scrollUpDownButton(where, parent, line, updown)</u>	81
6.63	<u>buildTiles(appNum, appName)</u>	81
6.64	<u>alarmheaderReset()</u>	81
6.65	<u>buildAlarmHeader(appNum, appName, addAnimationsJobs, alarmTiles, neededforAlarmPage, tile, num)</u>	81
6.66	<u>getDetailsOverlay()</u>	81
6.67	<u>onRestAlarmReady(motherTable)</u>	81
6.68	<u>doFilter(filters, listelement, motherTable)</u>	81












































6.69	<u>resizeDetail(self)</u>	81
6.70	<u>reloadTables()</u>	81
6.71	<u>preparePrivateTabs(myval, mykey, toggleme, myText)</u>	81
6.72	<u>onPrivateReady(switcher)</u>	81
6.73	<u>buildFilterOverlay(filterElements, tdName, parent, self)</u>	82
6.74	<u>resetter(addClass, tab, parent, self)</u>	82
6.75	<u>filterMyList(self, tabClass, filterval, motherTable)</u>	82
6.76	<u>removeFilter(self)</u>	82
6.77	<u>unique(array)</u>	82
6.78	<u>closeDetail(self)</u>	82
6.79	<u>saveHeadline(self)</u>	82
6.80	<u>togglePrivPub(self, myid)</u>	82
6.81	<u>getPrivateTabs(preparent, forOverlay, toggleme, private)</u>	82
6.82	<u>getPrivateTabFooter()</u>	82
6.83	<u>actionForSave(e)</u>	82
6.84	<u>actionForCancel(e)</u>	82
6.85	<u>loadIframe(e, alarmheader)</u>	82
6.86	<u>onSiteProperties(msg)</u>	83
6.87	<u>activateButton()</u>	83
6.88	<u>terminateActiveAlarmClient()</u>	83
6.89	<u>addTrendPage(parent, myPlc, showControl)</u>	83
6.90	<u>buildAlarmList()</u>	83
6.91	<u>changeActive(self, showme)</u>	83
6.92	<u>listen_again()</u>	83

1 Serverstruktur

Hier eine Abbildung, wie die Frontend-Daten auf dem Server liegen.

1.1 Ordnerstruktur



- ▼  components
 -  ui-router
- >  config_alt
- ▼  contents
 - >  Applcation
 - >  APT
 - >  Content
 - >  Conveyor
 - >  Design Elements
 - >  Diagnosis
 - >  hmi
 - >  Process
 - >  PT_DS1
 - >  SmartHambach
 - >  Supervisory
 - >  VWDresden
- ▼  css
 -  font
- ▼  directives
 -  contentsection
 -  contentsectionPlant
 -  gridelement
 -  gridelementPlant
 -  json
 -  mainmenu
 -  navigationBar
 -  statusBar
 -  wizard
 -  wizardRelated
 -  wizardSolutions
 -  wizardTrending
- ▼  factories
 -  dataFactory
- ▼  filters
 -  messageFilter
 -  minutesToHHmm
 -  slice
-  images
- ▼  js
 - >  amstockchart
 - >  js
 - >  TrendPage
 -  keyboard

Das Dürr HMI Framework besteht aus 3 Hauptteilen, dem Header, dem Hauptmenü und dem Content-Bereich. Der Header-Bereich teilt sich auf in Kunden-Logo, Kopfzeile, Status- und Alarmzeile dem Software-Bereich und dem Dürr-Logo. Der Content-Bereich teilt sich in Inhaltsbereich und Rechter Bereich.

Als Datenlieferanten stehen die emos.js und json-Files zur Verfügung. Das Dürr HMI Framework kann für jeden Computer individuell eingerichtet werden. Um ein individuelles Erscheinungsbild darzustellen, werden 2 Parameter bei Start der Applikation mitgegeben. Dies sind zum einen der Stationsname (z.b. sn=Trockner_1), der auf der geladenen Applikation in der Kopfzeile auf der rechten Seite angezeigt wird und die für diese Station angelegte json-Datei (z.b. app=diagnosis).

2.1 Daten per json-Datei

Die für den individuellen Aufbau der Applikation benötigten json-Datei beinhaltet derzeit die Informationen, welche Sprachen zur Verfügung stehen, welche Applikationen im Hauptmenü angezeigt werden, die dazu gehörigen Anlagen, die als Kacheln im Content-Bereich angezeigt werden und die Informationen der zu der Applikation gehörenden Alarmbuttons, die in der Status- und Alarmzeile platziert sind. Auch kann hier die Start-Configuration angegeben werden, welches Bild von welcher Anlage aus welcher Applikation gezeigt werden soll, wenn ein User auf das Kundenlogo klickt.

Beispielaufbau der diagnosis.json-Datei, hinter // folgt ein Kommentar, gehört nicht in eine json-Datei:

```
{
  "sidebuttons": [
    {
      "name": "Process", // Name der Applikation
      "textId": "T00_0047", // Text_Id für die Sprachdarstellung
      "stationname": "EcoScreen", // Produktname
      "iconclass": "process", // Klassenname für das benötigte Icon
      "alarm": "Benchmark_PLC1.Boolean.0403", // für die Alarmanzeige im
      Button
      "tiles": [ // Auflistung der zugehörigen Anlagen
        {
          "name": "Diagnosis", //Anlagenname
          "iconclass": "booth", Klassenname für das benötigte Icon
          "link": "contents/Diagnosis/Diagnosis_Window_Test.htm", // link für
          Klick-Aktion
          "textId": "PT_PTP_xxxxx", //Text_Id für die Sprachdarstellung
          "alarm": "Benchmark_PLC1.Boolean.0401",
          "alarm_security": "Benchmark_PLC1.Boolean.0411",
          "IO_Auto": "Benchmark_PLC1.Boolean.0421",
          "IO_SSEN": "Benchmark_PLC1.Boolean.0431",
          "IO_CSON": "Benchmark_PLC1.Boolean.0441",
          "priority": 0
        } // weitere Anlagen ...
      ],
      "alarm_buttons": [
        {
          "name": "Handbetrieb",
          "iconclass": "automatic",
          "link": "",
          "alarm": "Benchmark_PLC1.Boolean.0403",
          "alarm_type": "IO",
          "alarm_id": "IO_Auto"
        },

```

```

        {
            "name": "Sicherheit",
            "iconclass": "safety",
            "link": "",
            "alarm": "Benchmark_PLC1.Boolean.0401",
            "alarm_type": "Alert",
            "alarm_id": "alarmsecurity"
        } // weitere Buttons ...
    ]
} // weitere Applikationen ...
],
"language": {
    "70": {
        "short": "DEU",
        "long": "Deutsch"
    } // weitere Sprachen ...
},
"start":{
    "application": "Process",
    "plant":"Pretreatment",
    "picture":"PE013_Degrease_3.htm"
}
}

```

2.2 Javascript/jQuery-Funktionen für den Seitenaufbau

Beim Laden der Applikation werden zuerst einige Event-Listener gesetzt:

„load“, „language“, „authentication“ und „UserRight“ wobei letzterer noch nichts bewirkt.

„load“ - startet die Connection zur emos.js/WebSocket

„language“ - setzt die Sprache und reagiert ggf. auf eine Sprachänderung

„authentication“ - überwacht den Login/-out Prozess

Dann werden die URL-Parameter ausgelesen um im nächsten Schritt mittels Ajax die benötigte json-Datei zu laden und in der Variablen „result“ bereitzustellen.

Um das Hauptmenü, bestehend aus den sog. Sidebuttons, aufzubauen, wird eine \$.each-Schleife mit den Werten aus „result.sidebuttons“ durchlaufen um zum einen eine unsortierte Liste und zum anderen die benötigten Alarmanimationen anzulegen. Jeder Menüpunkt, Sidebutton, ist ein Listenelement:

```

"<li class='cats'><span class='emosbutton " + val.iconclass + "'><span
class='' id='" + id + "'></span><p data-num='" + d_num + "'
class='buttontext' id='" + val.textId + "'>" + val.name +
"</p></span></li><span class='trenner'></span>".

```

Jeder Sidebutton beinhaltet ein Alarm/Warning-Icon welches anzeigt, ob es in einem oder mehreren der Applikation zugehörigen Anlagen Alarime oder Warnungen gibt. Hierfür werden in der Funktion

```
getAnimationData(val.name, val.tiles);
```

sämtliche dafür benötigten Daten gesammelt

```

$.each(tile, function (key, val) {
    io_auto.push(val.IO_Auto);

```



```

        alarmsecurity.push(val.alarm_security);
        io_ssen.push(val.IO_SSEN);
        io_cson.push(val.IO_CSON);
        alarmgroup.push(val.alarm);
    });
    aniData[name] = {
        'alarmsecurity': alarmsecurity,
        'io_auto': io_auto,
        'io_ssen': io_ssen,
        'io_cson': io_cson,
        'alarmgroup': alarmgroup
    };

```

und mit

```

alarmAnimations1.push(new emosWS.HTMLFaultWarning({
    "id": id,
    "alarmGroup": aniData[val.name].alarmgroup
}));

```

in ein weiteres Array geschrieben, damit sämtliche auf der Applikation sichtbaren Alarmer im gleichen Rhythmus blinken. `emosWS.HTMLFaultWarning` sorgt nun dafür, dass Alarmer und Warungen angezeigt werden.

Nachdem die Schleife durchlaufen ist, erstellen wir die UL mittels der kleinen Funktion `ulMaker('maincats', items, '#left')`; // ID, Elemente, Parent

Damit auch immer der Richtige Applikationsname in der eingestellten Sprache angezeigt wird, können wir nun, da jedes Button-Element erstellt und durch eine ID auffindbar ist, mittels der Funktion

```

results.sidebuttons.forEach(function (val) {
    var text = document.getElementById(val.textId);
    if (val.textId) {
        text.appName = val.name;
        emosWS.sendAdviseText(val.textId, "name", function (msg) {
            text.innerHTML = msg.value;
        });
    }
});

```

auch dafür sorgen.

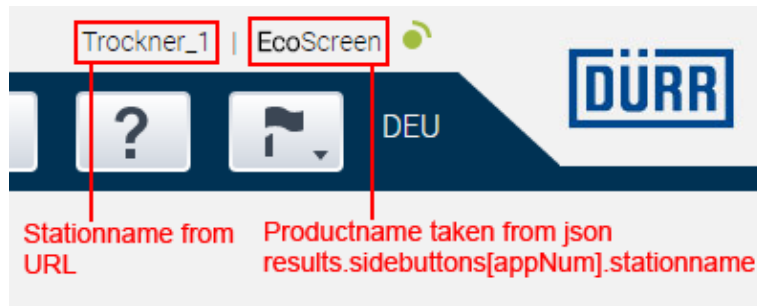
Der oberste Sidebutton in der json-Datei ist zusätzlich noch die Einstellung für den Start, soll heißen, diese Applikation/Sidebutton ist als aktiv gekennzeichnet und die dazugehörigen Anlagen(Tiles) werden im Content-Bereich dargestellt. Um diese Kacheln aufzubauen nutzen wir die Funktion

[6.63.buildTiles\(appNum, appName\)](#)

die auch bei einem Klick auf einen der Sidebutton aufgerufen wird(dann aber mit

anderen Parametern).

Die Funktion buildTiles(die nochmals überarbeitet werden muss/kann, anm. d. Red), sorgt für die Aufbereitung der Daten, die zu der gewählten Applikation gehören. Sie schreibt den Stationsnamen, der aus der Url extrahiert wurde und den Produktnamen, der aus result.sidebuttons gelesen wird, in die Kopfzeile auf die rechte Seite



und setzt den aktuellen Menüpunkt auf aktiv.

Sämtliche zugehörige Anlagen werden auch im Alarmoverlay angezeigt(Beschreibung unten), deshalb auch einige Berechnungen an dieser Stelle. Je nach Bildschirmgröße werden diese Anlagen dort ein- oder mehrspaltig angezeigt, dies berechnen wie durch

```
var columnContent = Math.floor($('#precenter').height() / 80);  
    var numOfCols = Math.ceil(eval(appToLoad).length /  
columnContent);  
    for (var i = 0; i < numOfCols; i++) {  
        $('<div/>', {  
            "id": "column_" + i,  
            "class": "column"  
        }).appendTo('#alarmpage_content');  
    };
```

und erstellen dann die entsprechenden Spalten im Overlay.

Nun gehen wir wieder in eine \$.each()-Schleife, dieses mal für sämtliche Anlagen und erstellen a) Listenelemente für die im Content-Bereich gezeigten Kacheln

```
cars.push("<li data-link='" + val.link + "' class='tile " +  
val.iconclass + "'><span class='tilename'" + val.name + "</span><span  
class=''" + id + "'><span class='warn_image'" + "</span></span></li>");
```

und b) Listenelemente für die Anlagen im Alarmoverlay

```
alarmTiles.push("<li id='" + val.name + "_" + id + "' data-link='" +  
val.link + "' class='firsttile emosbutton " + val.iconclass + "'><input  
class='plantcheckbox' type='checkbox' name='plants' value='salami'" + "<span  
class='tilename'" + val.textId + "<br>" + val.name + "</span><span  
class=''" + apageId + "'><span  
class='warn_image'" + "</span></span></li>");
```

und c) springen wir in die

[6.65.buildAlarmHeader\(appNum, appName, addAnimationsJobs, alarmTiles,](#)

[neededforAlarmPage, tile, num](#))-Funktion.

Die Anlagen-Liste im AlarmOverlay erhält die gleichen Alarm-Icons, die oben in der Status- und Alarmzeile zu sehen sind. Deshalb wird diese Funktion auch 2x angesprochen. Hier wird wiederum eine \$.each-Schleife durchlaufen, dieses mal mit den in der json zu der Applikation zugeordneten Alarmbuttons:

```
alarmToLoad = "results.sidebuttons[" + appNum + "].alarm_buttons";
```

um zwei Listen zu erstellen; eine für die Status- und Alarmzeile und eine für das AlarmOverlay:

```
alarms.push("<li class='emosbutton " + val.iconclass + "'><span  
class=' ' id=' " + id + "'></span></li>");  
  
if (alarmTiles)  
    alarmTiles.push("<li class='emosbutton " + val.iconclass +  
    "'><span class=' ' id=' " + alarmpage_id + "'></span></li>");
```

Natürlich benötigen wir noch die Informationen, welche Art Alarm in den Elementen angezeigt werden soll:

```
switch (val.alarm_id) {  
    case 'IO_SSEN':  
        tmpalarm = tile ? tile.IO_SSEN :  
aniData[appName].io_ssen;  
        break;  
    case 'IO_Auto':  
        tmpalarm = tile ? tile.IO_Auto :  
aniData[appName].io_auto;  
        break;  
    case 'alarmsecurity':  
        tmpalarm = tile ? tile.alarm_security :  
aniData[appName].alarmsecurity;  
        break;  
    case 'IO_CSON':  
        tmpalarm = tile ? tile.IO_CSON :  
aniData[appName].io_cson;  
        break;  
    case 'alarmgroup':  
        tmpalarm = tile ? tile.alarm :  
aniData[appName].alarmgroup;  
        break;  
}  
  
var tmp_id;  
if (neededforAlarmPage) {  
    tmp_id = alarmpage_id;  
} else {  
    tmp_id = id;
```

```

    }
    addAnimationsJobs.push(function () {
        if (val.alarm_type === "IO") {
            alarmAnimations.push(new emosWS.HTMLSwap({
                "id": tmp_id,
                "plctag": tmpalarm,
                "falseBlinking": false
            }));
        } else {
            alarmAnimations.push(new emosWS.HTMLFaultWarning({
                "id": tmp_id,
                "alarmGroup": tmpalarm
            }));
        }
    });
});

```

Damit haben wir nun die für den Alarmoverlay benötigten Daten gesammelt und können nun die Liste erstellen und in die berechnete Spalte positionieren:

```

    var columnNum = Math.floor(key / columnContent);
    ulMaker('alarm_solo', alarmTiles, '#column_'+ columnNum,
'alarm_solo_' + id);
    alarmTiles = [];

```

Wir benötigen aber auch noch einen Alarm für die Anlagen-Kachel und verlassen die Schleife:

```

    addAnimationsJobs.push(function () {
        alarmAnimations.push(new emosWS.HTMLFaultWarning({
            "id": id,
            "alarmGroup": val.alarm
        }));
    });
});

```

Aus dem Array mit den Listenelementen(cars) wird eine unsortierte Liste gemacht.

Da die Gesamtapplikation weitestgehend keine Scrollbalken aufweisen soll, es aber natürlich möglich sein kann, dass es mehr Anlagen-Kacheln gibt, als auf den Content-Bereich passen, erstellen wir 2 Scrollbuttons (up/down), die nur bei Bedarf angezeigt werden. Diese Buttons enthalten eine click-function zum hoch- bzw. runterscrollen der Anlagen-Kacheln. Sind mehr Kacheln als Platz vorhanden, so werden die Kacheln unterhalb eines bestimmten Punktes auf halbtransparent gesetzt. Kommen diese Kacheln beim Scrollen oberhalb dieses Punktes an, so werden sie komplett sichtbar.

```

$("<span/>", {
    "class": "scrollUp",
    "click": function() {

```

```

        var self = this;
        var liTop = $('#cars li:first').position().top;
        if($(this).hasClass('active')){
            $('#cars li').animate({
                "top": liTop + 380
            }, 300, function(){
                $(self).prev().addClass('active');
                $.each($('#cars li'), function () {
                    if($(this).position().top >= $(
('#center').height() - 170){
                        $(this).css('opacity', 0.5);
                    }
                });
                if($('#cars li:first').position().top >= 0){
                    $(self).removeClass('active');
                }
            });
        }
    }
    }).insertAfter('#cars');
    $("<span/>", {
        "class": "scrollDown",
        "click": function(){
            var self = this;
            var liTop = $('#cars li:first').position().top;
            if($(this).hasClass('active')){
                $('#cars li').animate({
                    "top": liTop - 380,
                    "opacity": 1
                }, 300, function(){
                    $(self).next().addClass('active');
                    $.each($('#cars li'), function () {
                        if($(this).position().top >= $(
('#center').height() - 170){
                            $(this).css('opacity', 0.5);
                        }
                    });
                });
                if($('#cars li:last').position().top + 170 <= $(
('#center').height())){
                    $(self).removeClass('active');
                }
            });
        }
    });
}

```

```

    }
  }).insertAfter('#cars');
$.each($('#cars li'), function () {
    if($(this).position().top >= $('#center').height() - 170){
        $(this).css('opacity', 0.5);
        $('#cars ~ .scrollUp, #cars ~ .scrollDown').show();
        $('#cars ~ .scrollDown').addClass('active');
    }
});

```

Damit sind die Kacheln im Content-Bereich gesetzt und es wurde auch für die Möglichkeit des Scrollens gesorgt. Aber die Status- und Alarmzeile und das Alarmoverlay sind noch nicht fertig. Wie oben bereits erwähnt, muss ich u.a. diese Funktion noch überarbeiten und verbessern. Ursprünglich war die Funktion `buildAlarmHeader()` dazu gedacht, nur die Alarm-Icons und deren Alarme/Warnungen in der Status- und Alarmzeile anzuzeigen. Da aber die Anlagen-Kacheln im AlarmOverlay eine ähnliche Anforderung erfüllen, wurde diese Funktionalität etwas erweitert. Nur wurden die Daten für den Header zwar angelegt, aber ich habe sie unterwegs verloren. Deshalb kommt an dieser Stelle erneut der Aufruf der Funktion, damit diese Daten dann auch angezeigt werden.



Zum Abschluss werden noch 4 Buttons erstellt, die im AlarmOverlay platziert werden. „Show details“, „reset“, „scrollLeft“ und „scrollRight“, deren Funktion weiter unten beschrieben wird.

Dieser Seitenaufbau, der nun fertig ist, bezieht sich auf unser Arbeitsbeispiel „Process“.

2.3 SASS/CSS

Kein HTML ohne CSS. Das Dürr HMI Framework nutzt natürlich auch CSS. Dies wird aber durch den Präprocessor SASS umgesetzt.

Der derzeitige Ansatz versucht eine thematische Trennung in Partialen, sodass man sich schnell zurecht finden kann. Die Gliederung sieht folgendermaßen aus:

```

styles_2.scss // sozusagen der Hauptteil, der alles andere importiert
_variables.scss // Variablen für Font, Farben, Schatten etc
_globals.scss // beinhaltet Placeholder und globalere Klassen
_mixins.scss // für wiederkehrende Muster
_header_2.scss // Styles für den Header-Bereich

```

_leftside.scss // Styles für das Hauptmenü
_center.scss // Styles für den Content-Bereich
_diagnose.scss // Styles für das Diagnose-Window

(Anmerkung: styles_2.scss und header_2.scss sind extra für die neue AlarmOverlayversion entstanden)

3 Header Funktionen

Nachfolgend werden die Funktionen/Use-Cases beschrieben, die sich im Header befinden.

3.1 Kunden-Logo / Dürr-Logo

Wie oben im Abschnitt json beschrieben, kann ein gewünschter Anlagenabschnitt gepflegt werden, damit dieser schnell durch einen Klick auf das Kundenlogo geladen wird. Nur reicht es nicht aus, die benötigte Html-Datei in den iFrame zu laden, es muss auch die dazugehörige Bread-Crumb-Navigation und die Status- und Alarmzeile angelegt werden. Die Funktion buildAlarmHeader wurde oben ja bereits beschrieben. Die Bread-Crumb-Navigation wird im Bereich Content-Bereich erleutert.

Ein Klick auf das Dürr-Logo lädt die allgemeine Startseite, also die in der json-Datei zu oberst stehende Applikation mit ihren Anlagen. Dazu wird ein einfacher Reload ausgeführt.

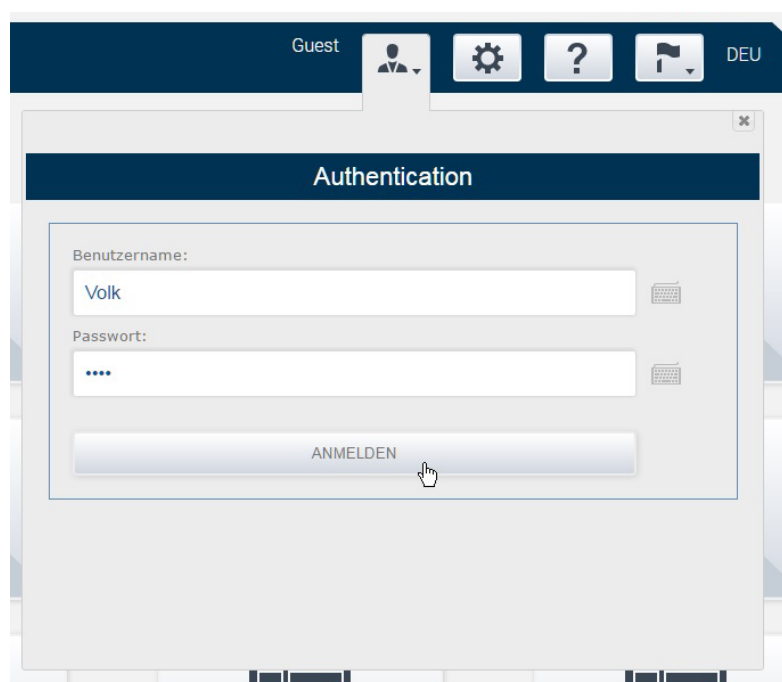
3.2 Log-In



Nach einem Klick auf den Log-In-Button senden wir eine Anfrage an das emos.js

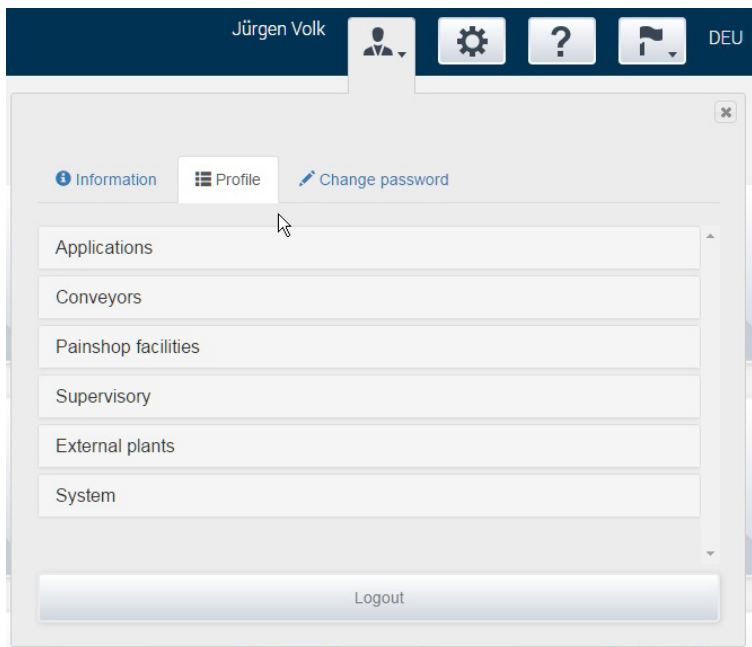
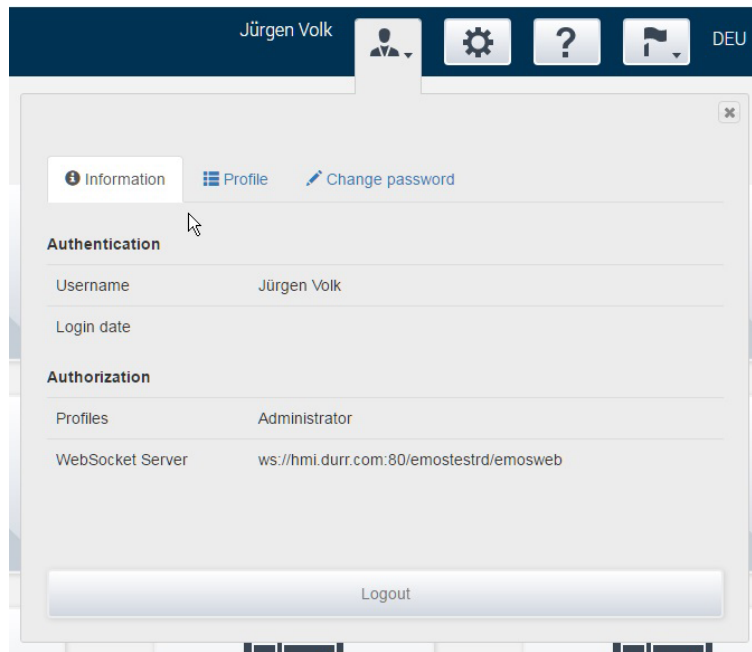
```
emosWS.login.au.showUI();
```

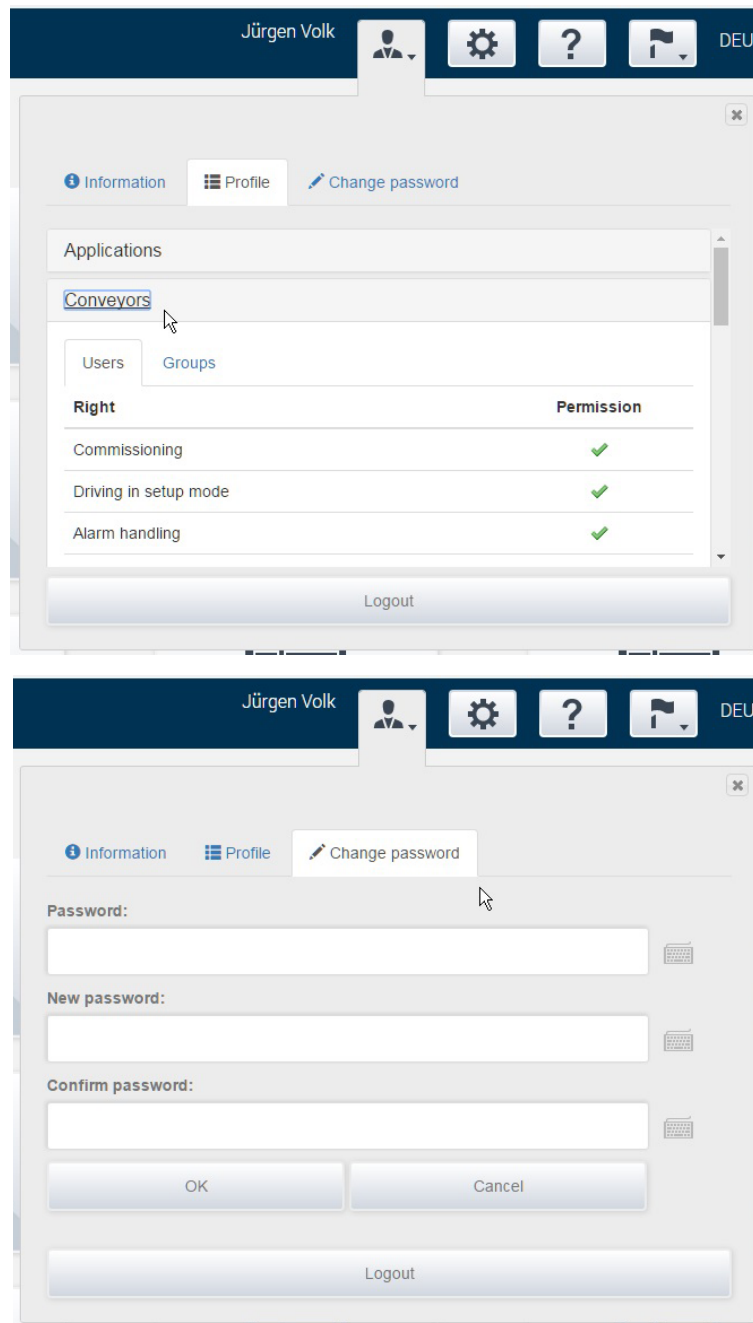
und es öffnet sich ein Overlay, in dem ein Log-In-Fenster vom CAS-Server geladen wird.

A screenshot of a web application interface. At the top is a dark blue header bar with the word 'Guest' on the left and several icons (user, settings, help, flag) on the right, followed by the text 'DEU'. Below the header is a light gray overlay window titled 'Authentication'. Inside this window, there are two input fields: 'Benutzername:' with the text 'Volk' and 'Passwort:' with masked characters '....'. Below these fields is a button labeled 'ANMELDEN'. A mouse cursor is pointing at the button. The background of the application is slightly dimmed.

Die Beschreibung zu diesem Fenster liegt hier noch nicht vor.

Nachdem ein Mitarbeiter seinen Nutzernamen und sein Passwort eingegeben und abgesendet hat, erhält die Applikation durch den EventListener("authentication") die Mitteilung durch den CAS-Server, dass die Anmeldung erfolgreich war(oder auch nicht) und den Benutzernamen. Dieser wird dann in den Softwarebereich links neben den Log-In-Button gesetzt. Ist der Mitarbeiter eingeloggt und klickt erneut auf den Log-In-Button, so öffnet sich erneut ein Overlay, in dem ein Fenster mit Userinformationen und auch mit einem Log-Out-Button geladen wird.





Auch dieses Fenster wird von CAS-Server geliefert.

3.3 Konfiguration



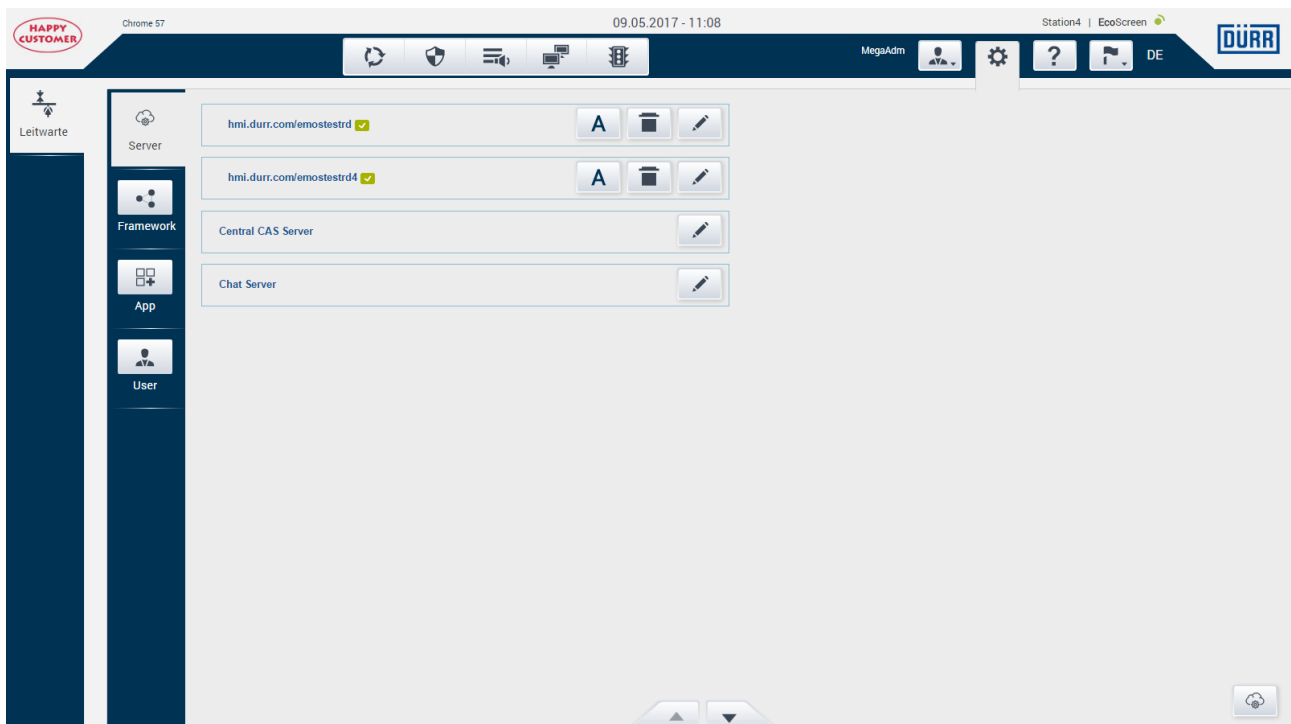
Im Konfigurationsbereich können allgemeine Einstellungen durch den Administrator vorgenommen werden, als auch individuelle durch den angemeldeten User. Der Administrator kann hier Server anlegen und bearbeiten, das Kundenlogo einpflegen, Sprachen auswählen, die den Usern dann zur Verfügung stehen, Chat und EcoDoku freigeben, default Sprache und Einheitensystem einstellen. Er kann Applikationen anlegen, bearbeiten und löschen

3.3.1 Server

Unter Einstellungen → Server werden sämtliche zur Verfügung stehende Server aufgelistet. Ein nicht angemeldeter User bzw. ein User ohne Admin-Rechten kann hier nur schauen, welche Server es gibt und wie deren Status ist. Zu jedem Server gibt es einen Link zu einer Seite mit tiefer gehenden Informationen zu dem gewählten Server. Ein Administrator kann an dieser Stelle einen neuen Server hinzufügen, einen vorhandenen Server bearbeiten oder löschen. Zum Aufbau dieser Seite wird die Funktion

[#6.2.GetServerList\(\)](#)

aufgerufen.



Möchte der Administrator einen neuen Server anlegen, sorgt der Klick auf den „new server“-Button dafür, dass die Funktion

[6.3.NewServer\(\)](#)

ausgeführt wird. Nach dem speichern des Servernamens sorgt dann die Funktion

[6.8.editServer\(dataFile, editserver, parent\)](#)

dafür, dass weitere benötigte Inhaltsfelder zur Verfügung stehen.

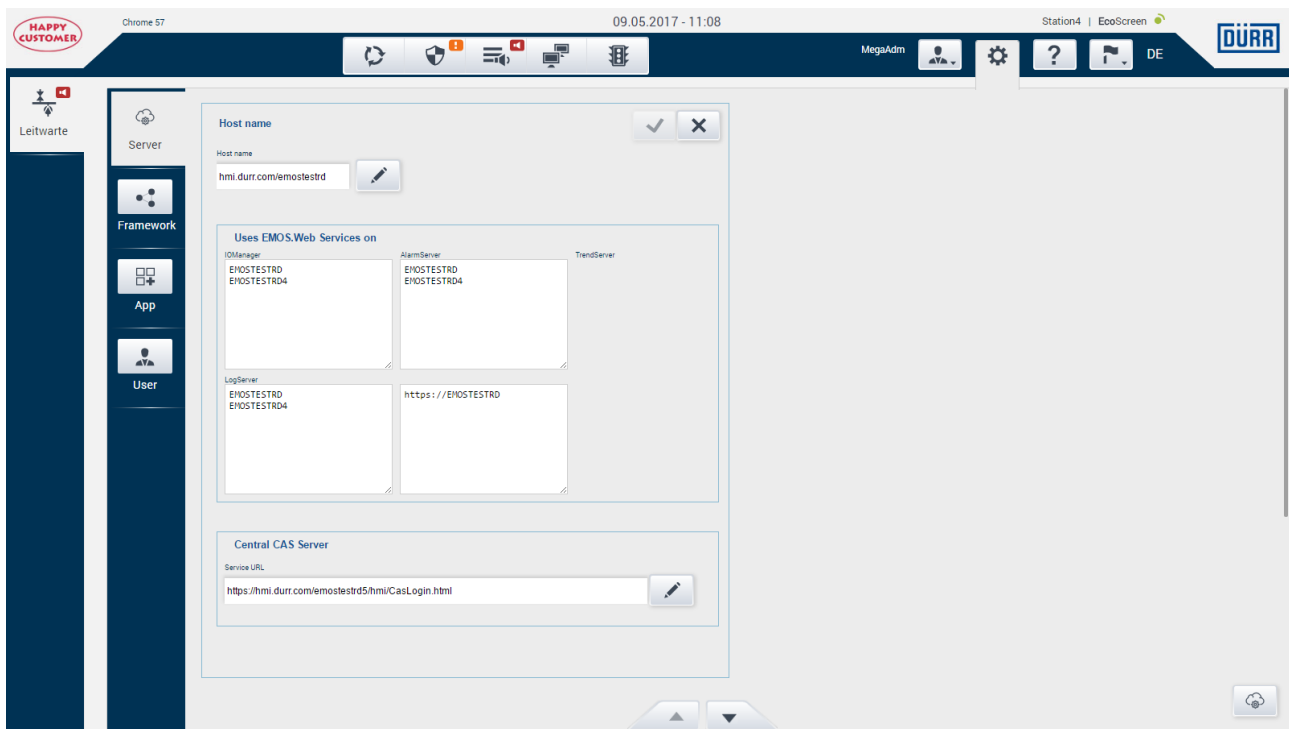
Möchte der Administrator nur einen Server bearbeiten, kommt nur die

[6.8.editServer\(dataFile, editserver, parent\)](#)

Funktion zum Einsatz, die aber auch in diesem Fall die

[6.3.NewServer\(\)](#)

Funktion aufruft, um den Servernamen in einem Textfeld anzuzeigen.



Wurden die Daten eingegeben, bzw. geändert, so wird der Speicher-Button aktiv und kann betätigt werden. Wird dieser geklickt, so wird die Funktion

[6.6.saveServer\(dataFile, self\)](#)

ausgeführt.

Klickt der Administrator in der Serverlist auf den delete-Button, um einen Server aus der Serverliste zu entfernen, so wird die Funktion

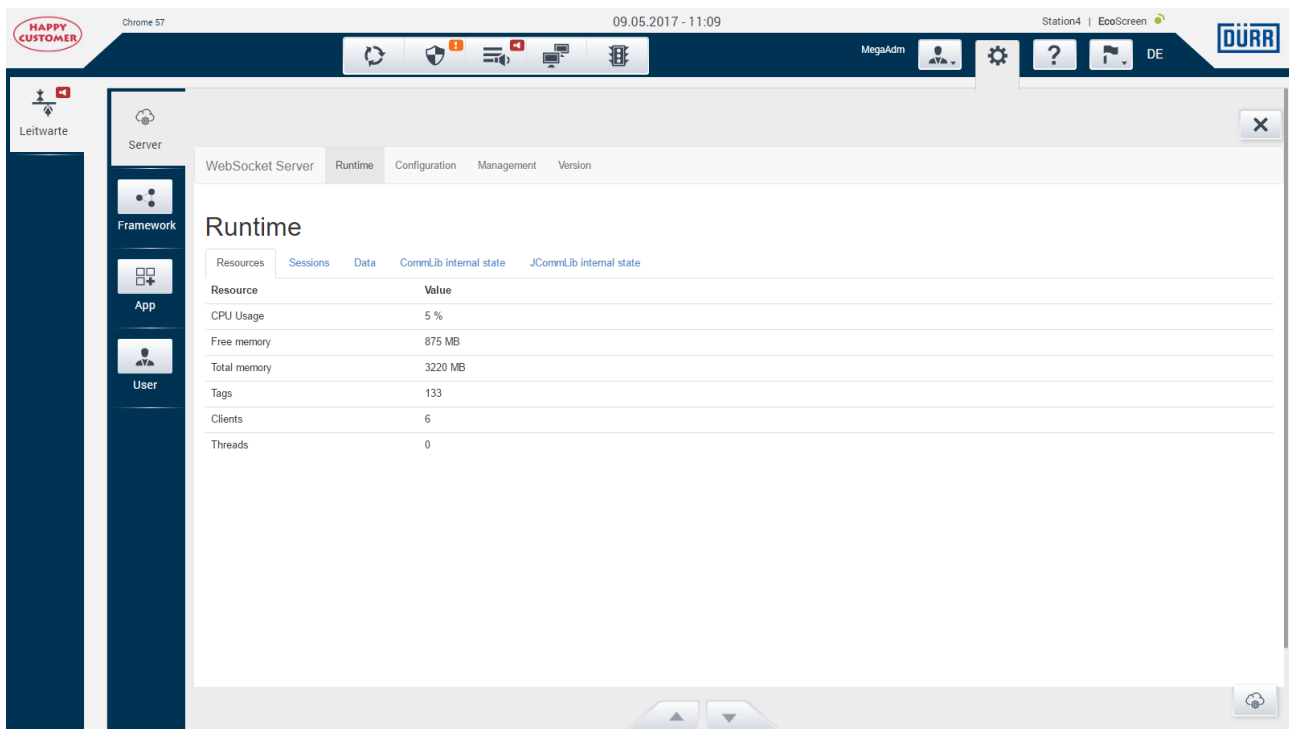
[6.7.deleteServerName\(dataFile\)](#)

ausgeführt.

Möchte der User mehr Informationen über einen Server haben, so sorgt ein Klick auf den „A“-Button dafür, dass die Funktion

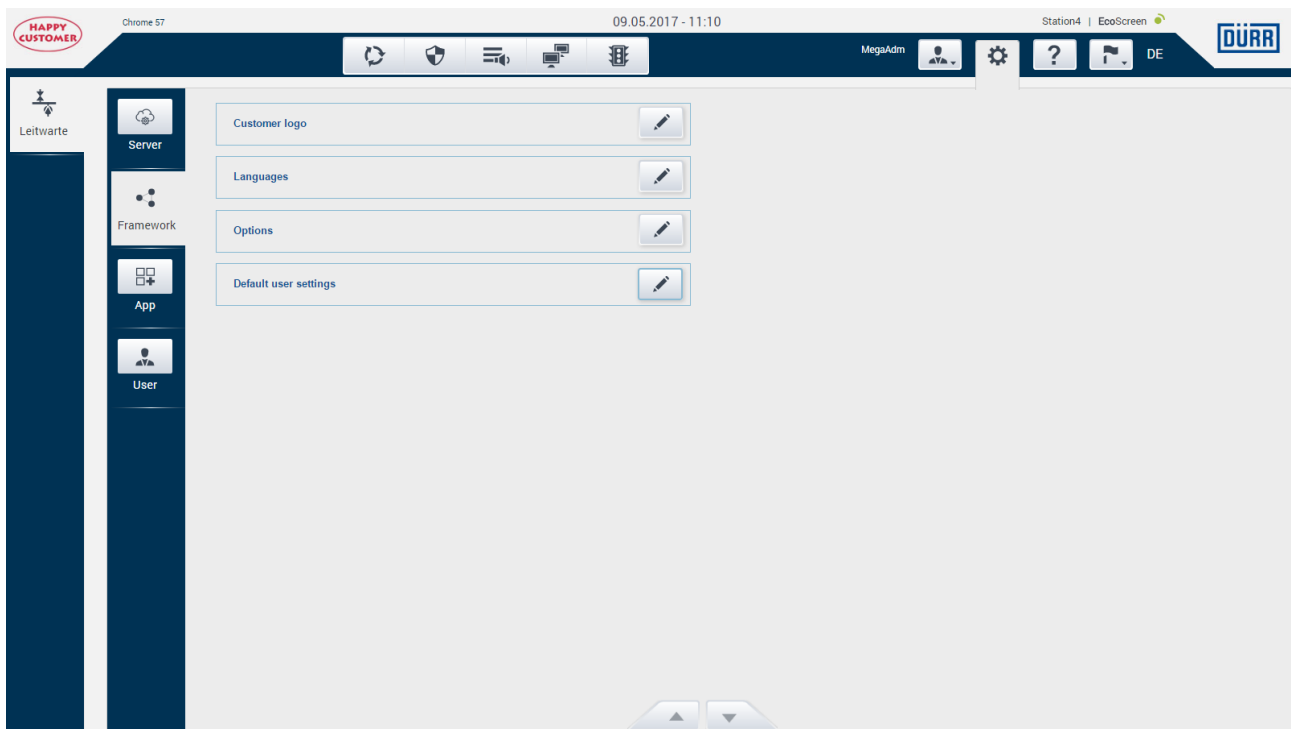
[6.9.showAdminpage\(dataFile, parent\)](#)

diese Informationen lädt.



3.3.2 Framework

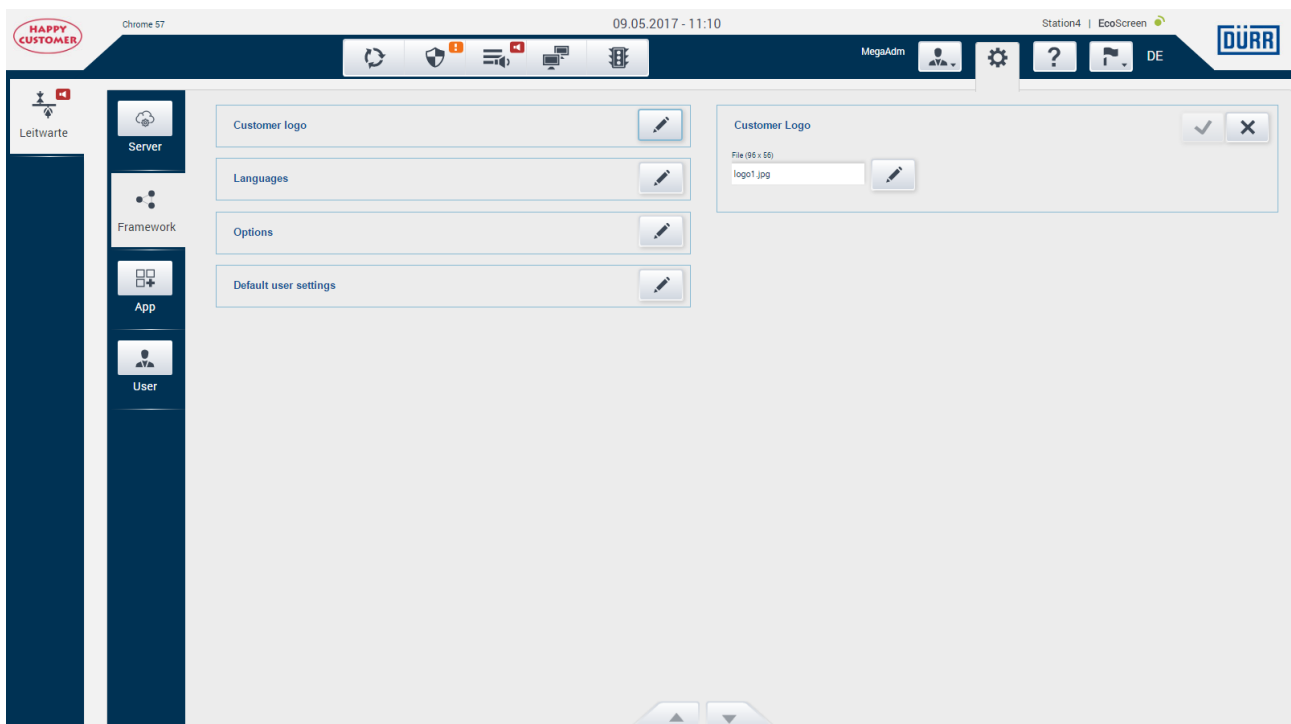
Unter dem Konfigurationspunkt Framework kann der Administrator das Kundenlogo einpflegen, Sprachen auswählen, die den Usern dann zur Verfügung stehen, Chat und EcoDoku freigeben, default Sprache und Einheitensystem einstellen.



Um ein neues Customer-Logo einzupflegen, kann der Administrator dies auf dem Rechner suchen und auf den Server hochladen. Die Funktion

[6.13.saveLogo\(\)](#)

sorgt dafür.

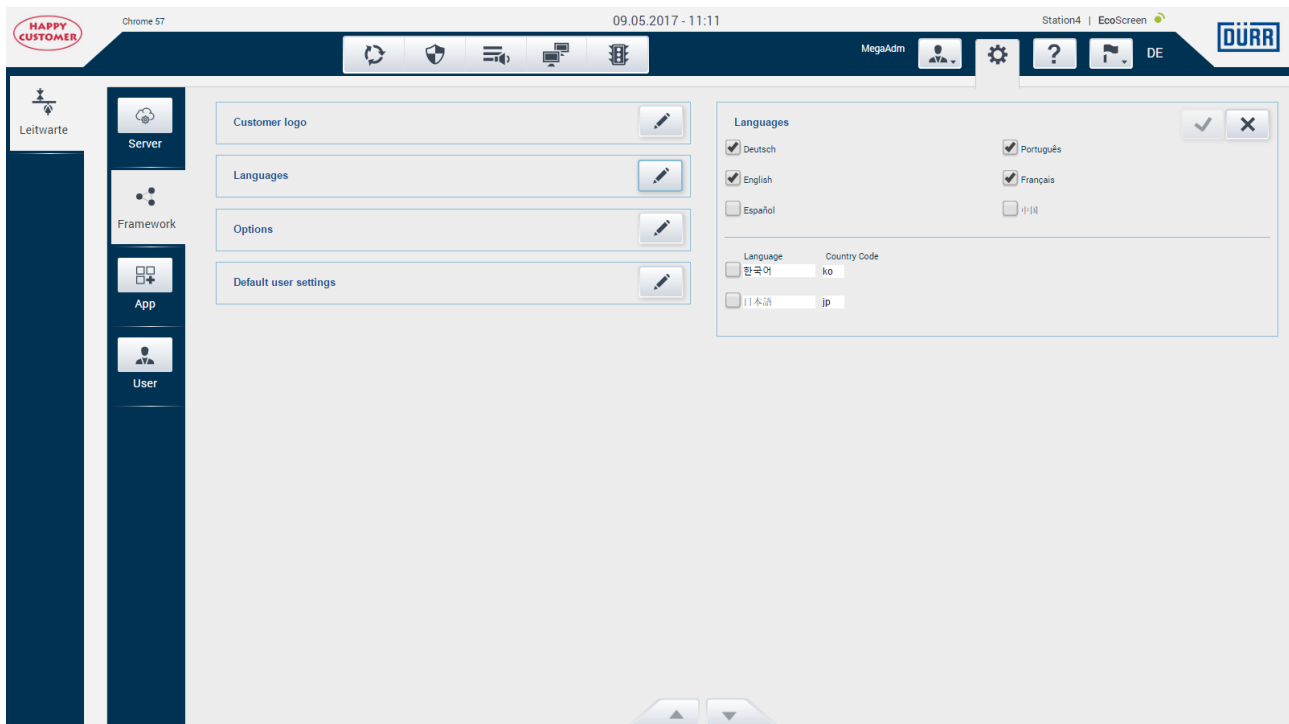


Wie bereits erwähnt, sorgt die Funktion

[6.11.getLanguageList\(\)](#)

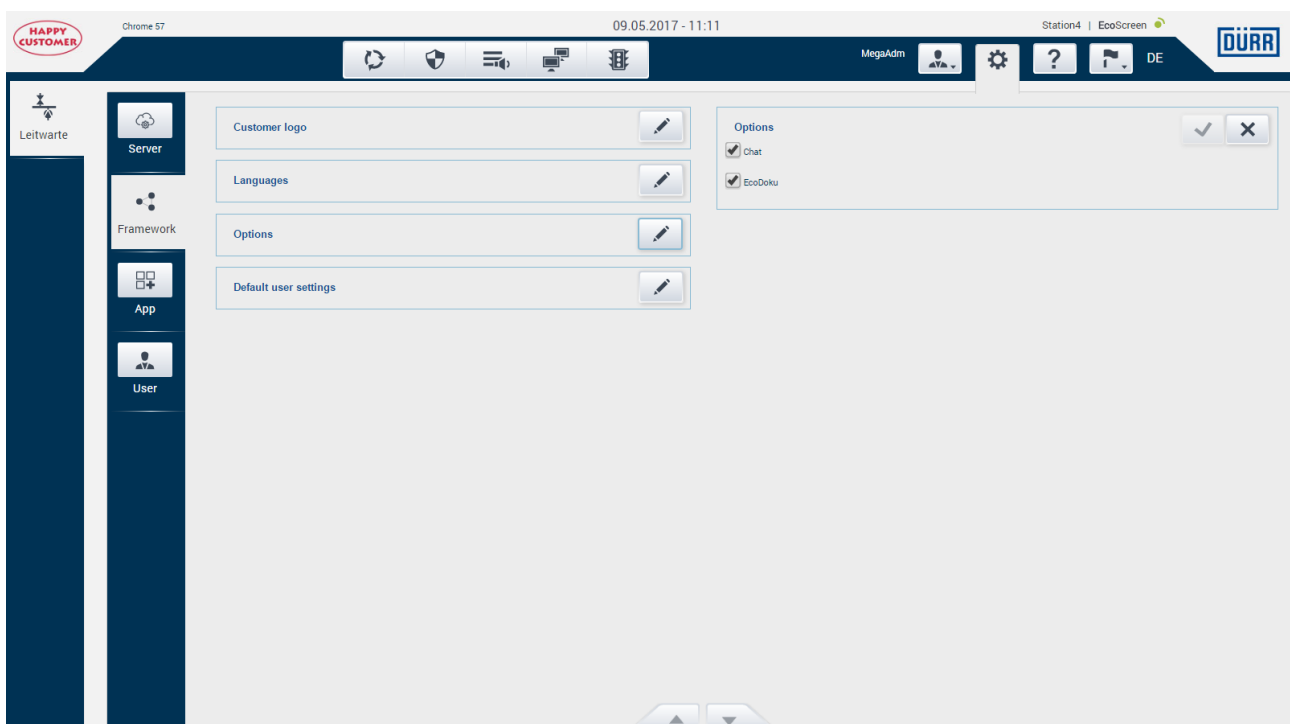
dafür, dass dem Administrator eine Liste der zur Verfügung stehenden Sprachen vorliegt. Unter dem Punkt „Languages“ kann er nun auswählen, welche Sprachen den Usern im

LanguageDropDown zur Auswahl stehen.



Nach der Auswahl und dem Klick auf Speichern wird die Funktion [6.12.saveLanguageList\(\)](#) ausgeführt.

Unter dem Punkt „Options“ kann der Administrator einen Chat und die EcoDoku freigeben. Zum speichern wird die Funktion [6.15.saveOptions\(\)](#) ausgeführt.

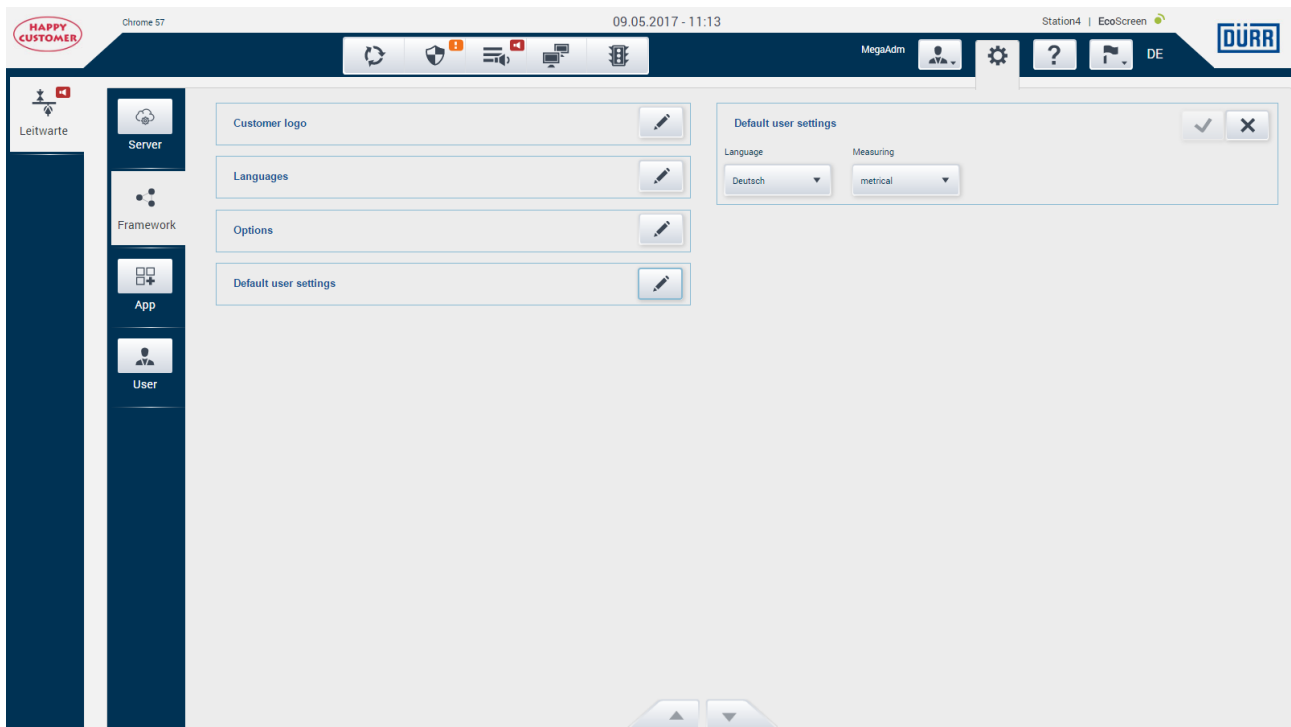


Unter dem Punkt „Default user settings“ kann der Administrator die default Sprache, also die Sprache, die automatisch angezeigt wird und das default metrische System einstellen.

Gespeichert wird das dann durch den Aufruf der

[6.14.saveSettings\(\)](#)

Funktion.

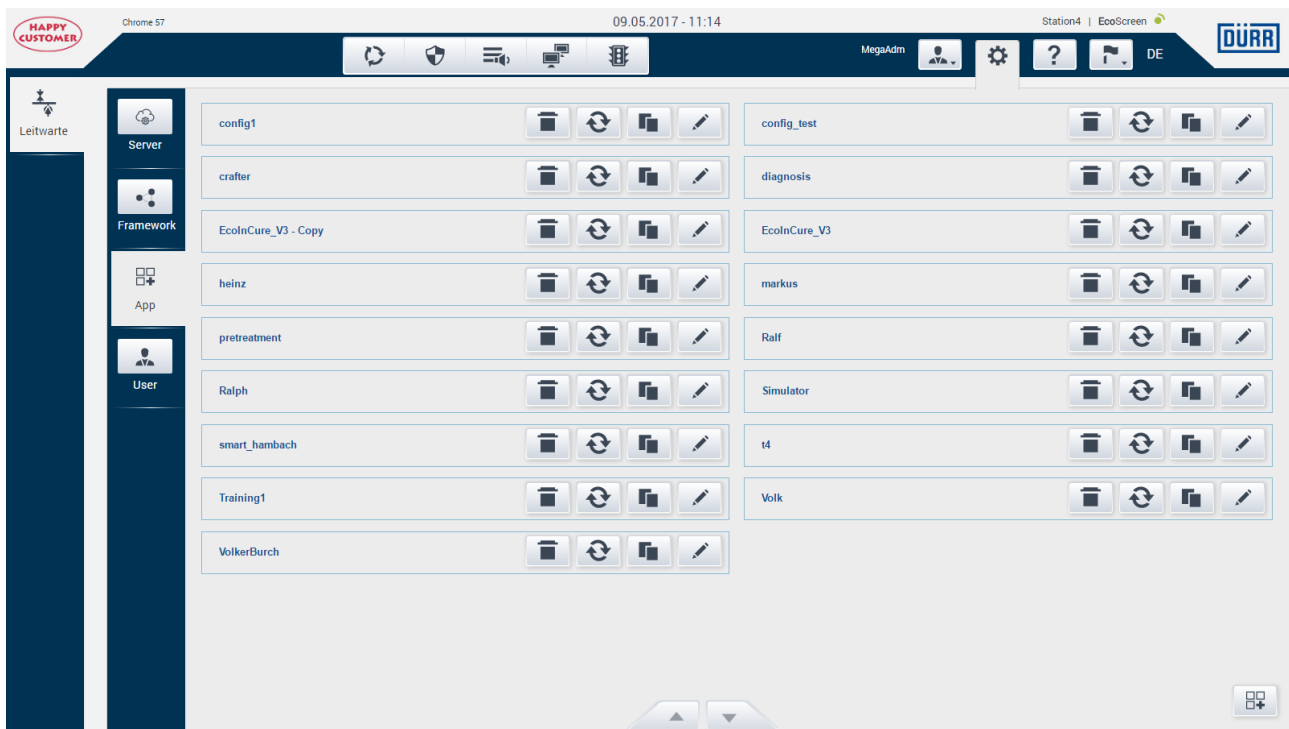


3.3.3 App

Unter dem Einstellungsmenüpunkt App findest sich eine Auflistung sämtlicher zur Verfügung stehender Applikationen die von allen Users mittels des load-Buttons geladen werden können, aber nur vom Administrator jeweils gelöscht, kopiert oder bearbeitet werden kann.

Aufgebaut wird diese Liste durch die Funktion

[6.24 buildAppList\(parent\)](#)



Klick auf den Button „delete App“ führt die Funktion

[6.44 getDeleteDialog\(filename\)](#)

aus. Damit wird dann die appname.json gelöscht und die

[6.24 buildAppList\(parent\)](#)

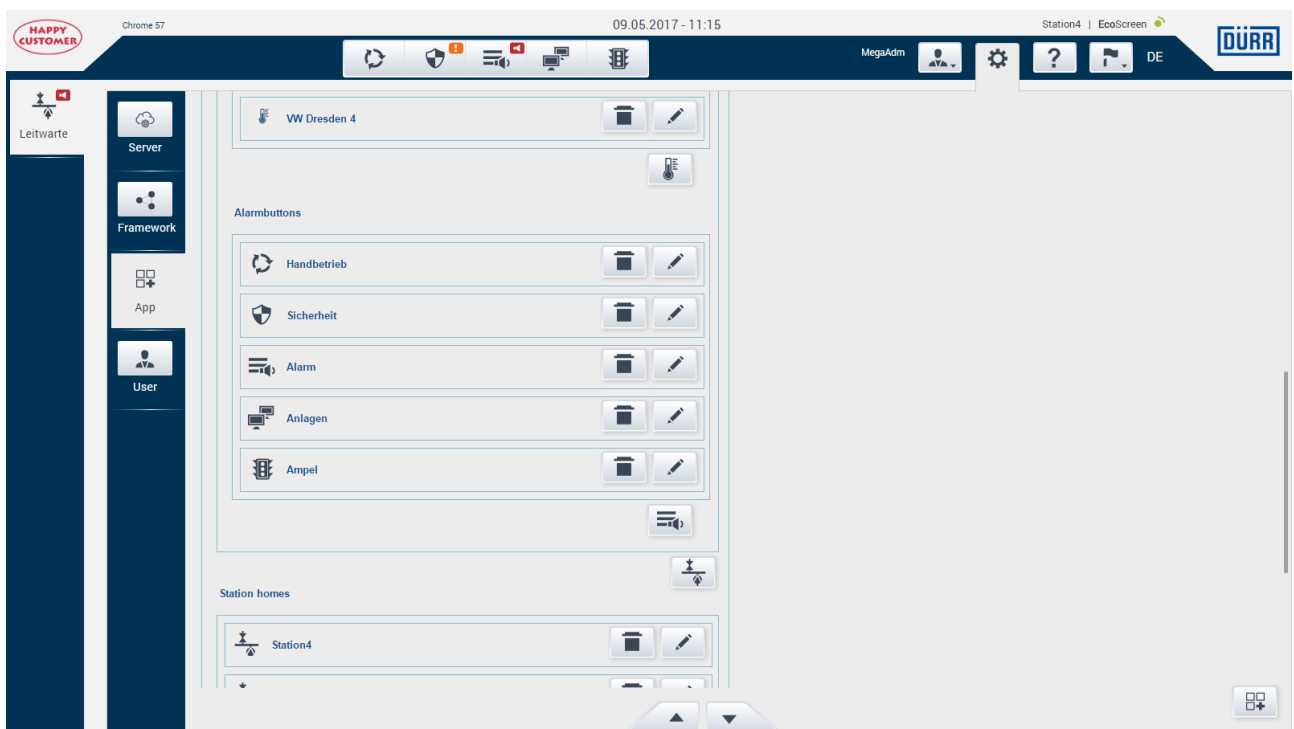
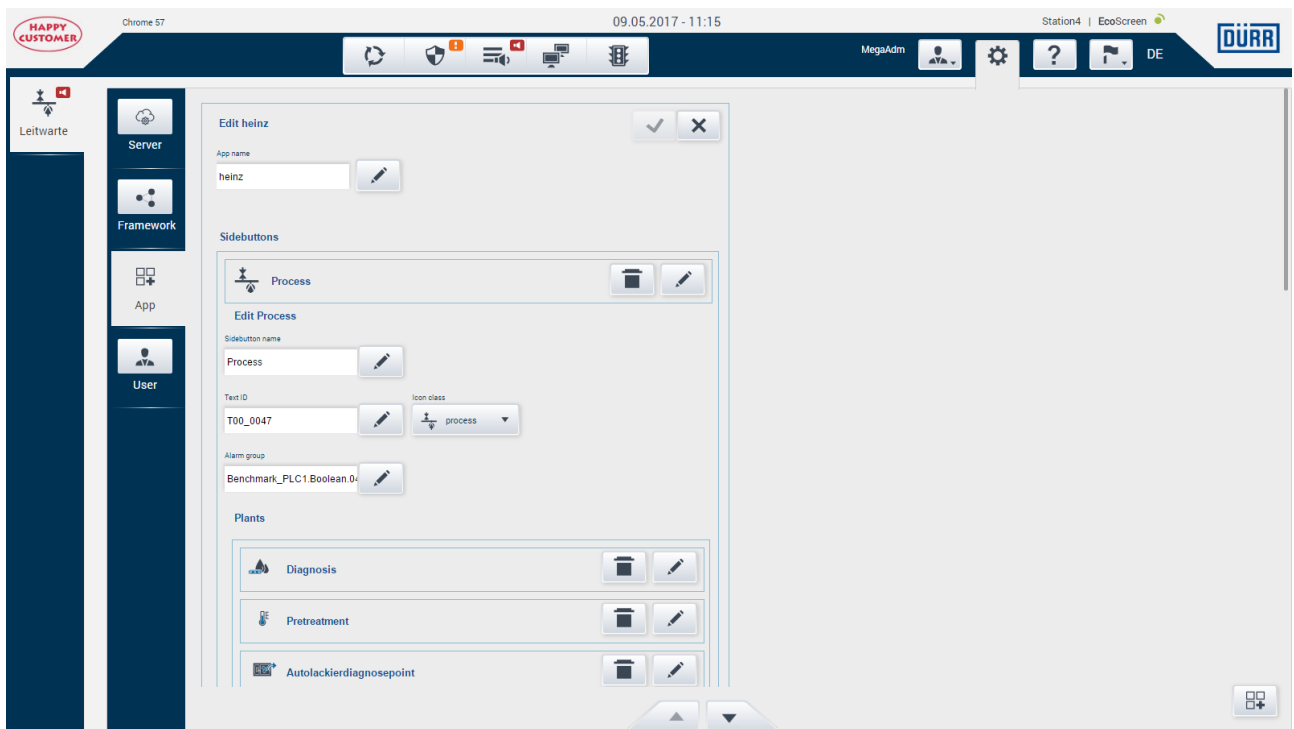
erneut ausgeführt.

Klick auf den „load“-Button lädt die gewünschte Applikation.

Bei Betätigung des „copy“-Buttons wird die Funktion

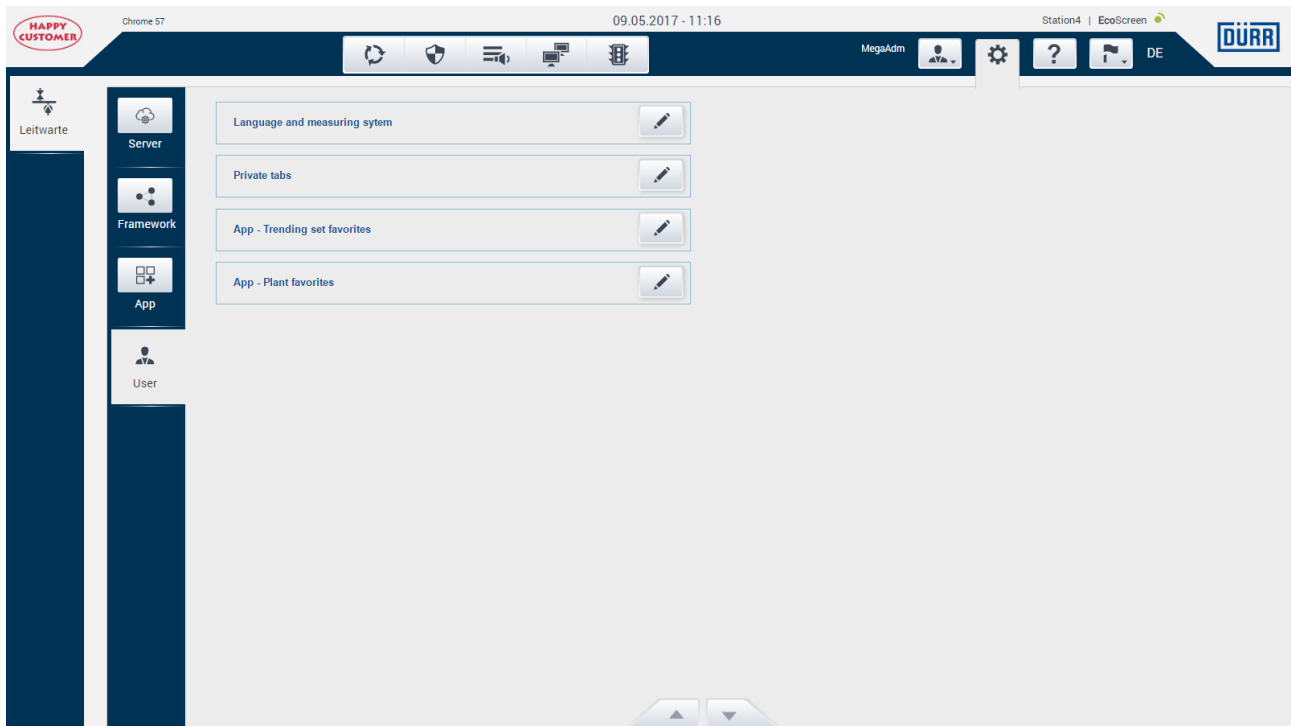
[6.35 editApp\(parent, dataFile, duplicateFile\)](#)

ausgeführt, ebenso wie nach dem Klick auf den „edit“-Button



3.3.4 User

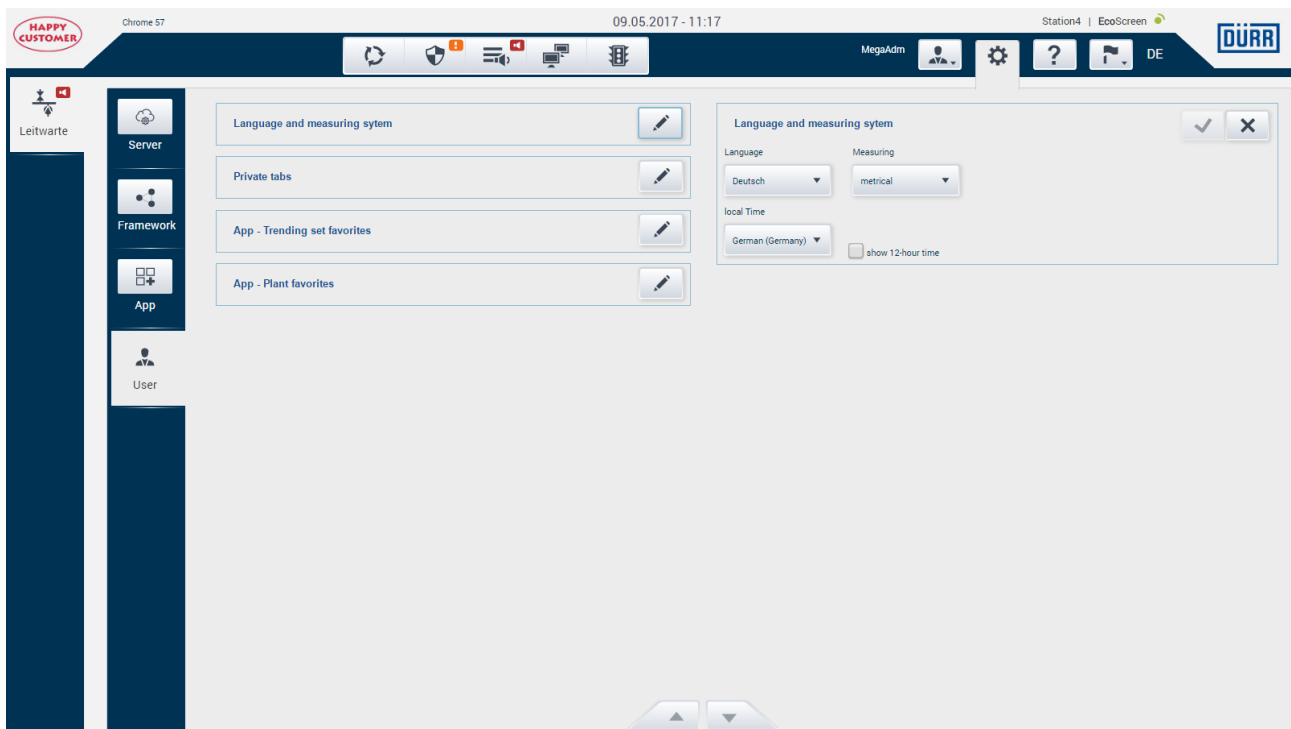
Jeder eingeloggte User hat die Möglichkeit, sich einige individuelle Einstellungen in seinem Profil zu speichern.



Unter dem Punkt „Language and measuring system“ kann sich der User seine bevorzugte Sprache, das Einheitensystem, die lokale Zeitanzeige und ob die Zeit in 12 oder 24 Stunden angezeigt werden soll, einstellen. Sind diese Einstellungen bereits im User-Profil vorhanden, werden diese direkt angezeigt. Gemachte Änderungen werden per

[6.16.saveUserLanguage\(\)](#)

gespeichert.



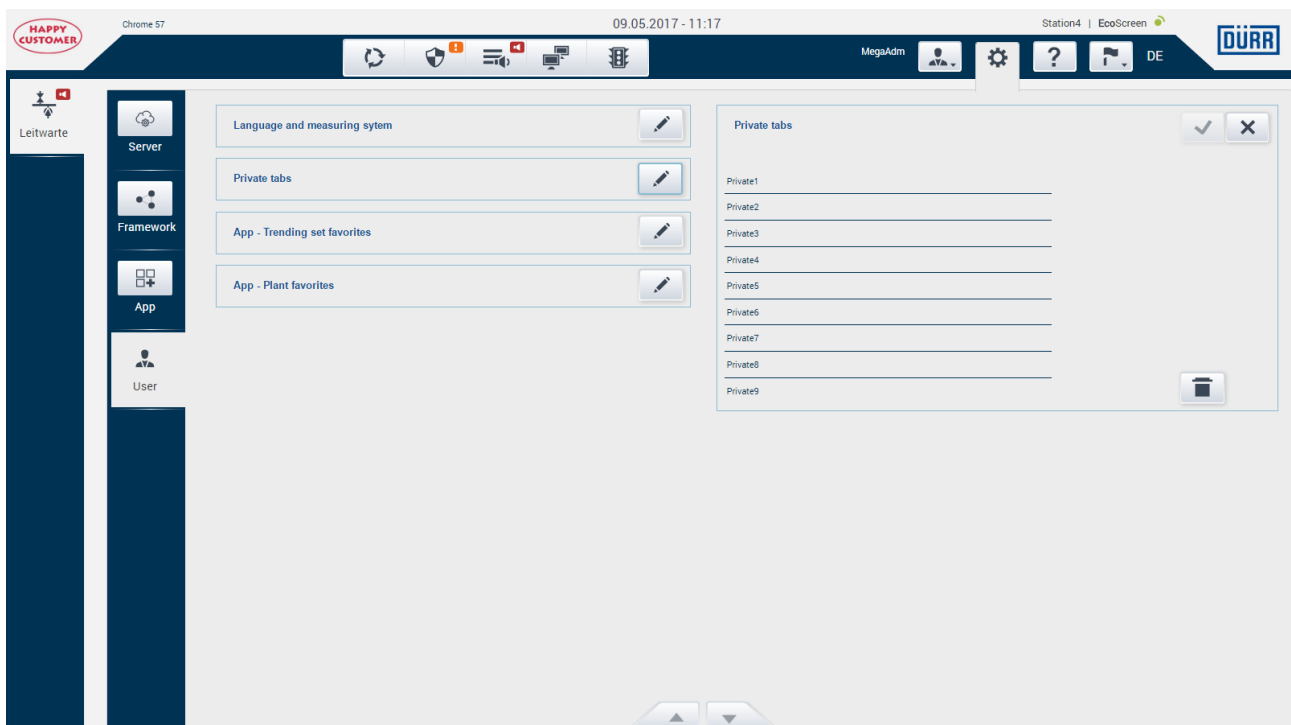
Der Punkt „Private tabs“ zeigt dem angemeldeten User eine Liste mit seinen individuell gestalteten Tab, falls diese vorhanden.

[6.17.getprivateTabList\(\)](#)

An dieser Stelle hat er auch die Möglichkeit, einen oder mehrere dieser Tabs zu löschen.

Wurde ein Tab gelöscht und dieses bestätigt, wird das neue Profil des Users auf dem Server gespeichert:

```
emosWS.writeData(writeJSONURL + myName + ".json", JSON.stringify(privateResults));
```

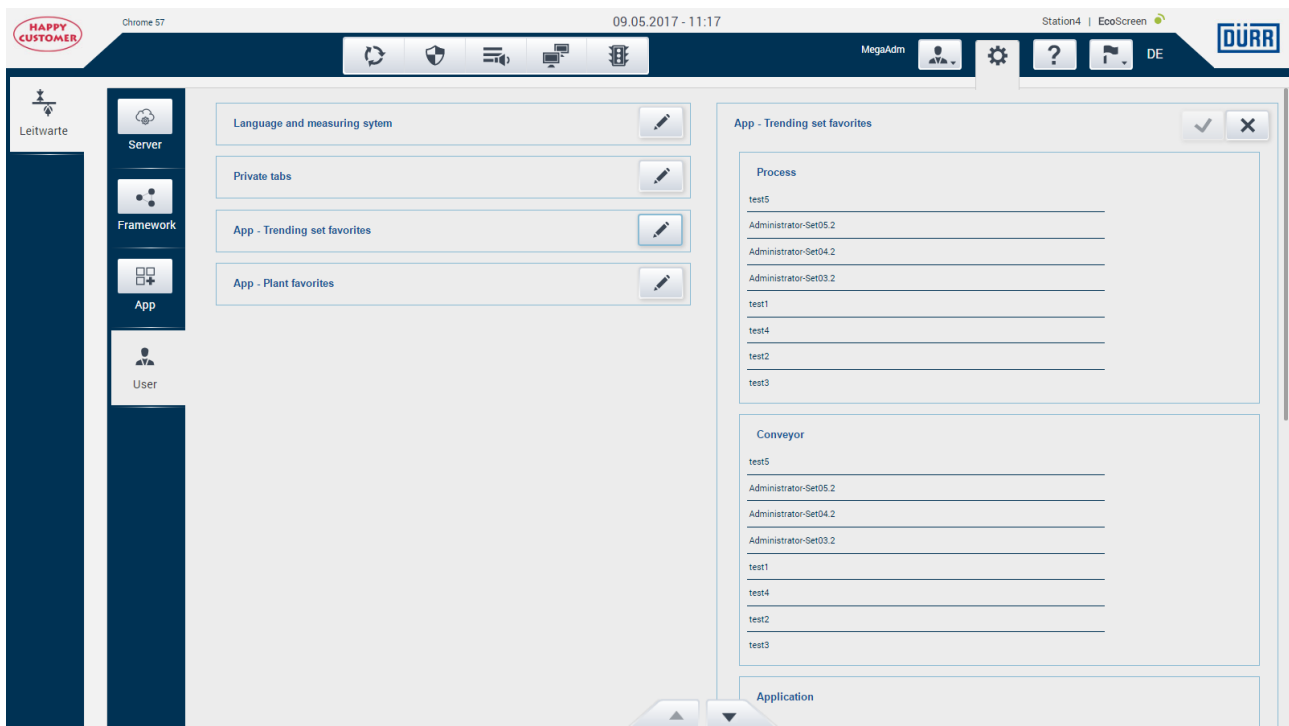


Im Bereich „App – Trending set favorites“ kann sich der User ein Trendingset auswählen, welches dann im Alarmoverlay automatisch angezeigt werden soll. Dafür wird eine Liste geladen, die sämtliche Public-Sets und seine private-Sets anzeigt.

[6.18.getTrendingSetList\(\)](#)

Nach einer erfolgten Auswahl und Bestätigung wird dies im Userprofil gespeichert.

[6.19.saveTrendingSetFavorites\(\)](#)

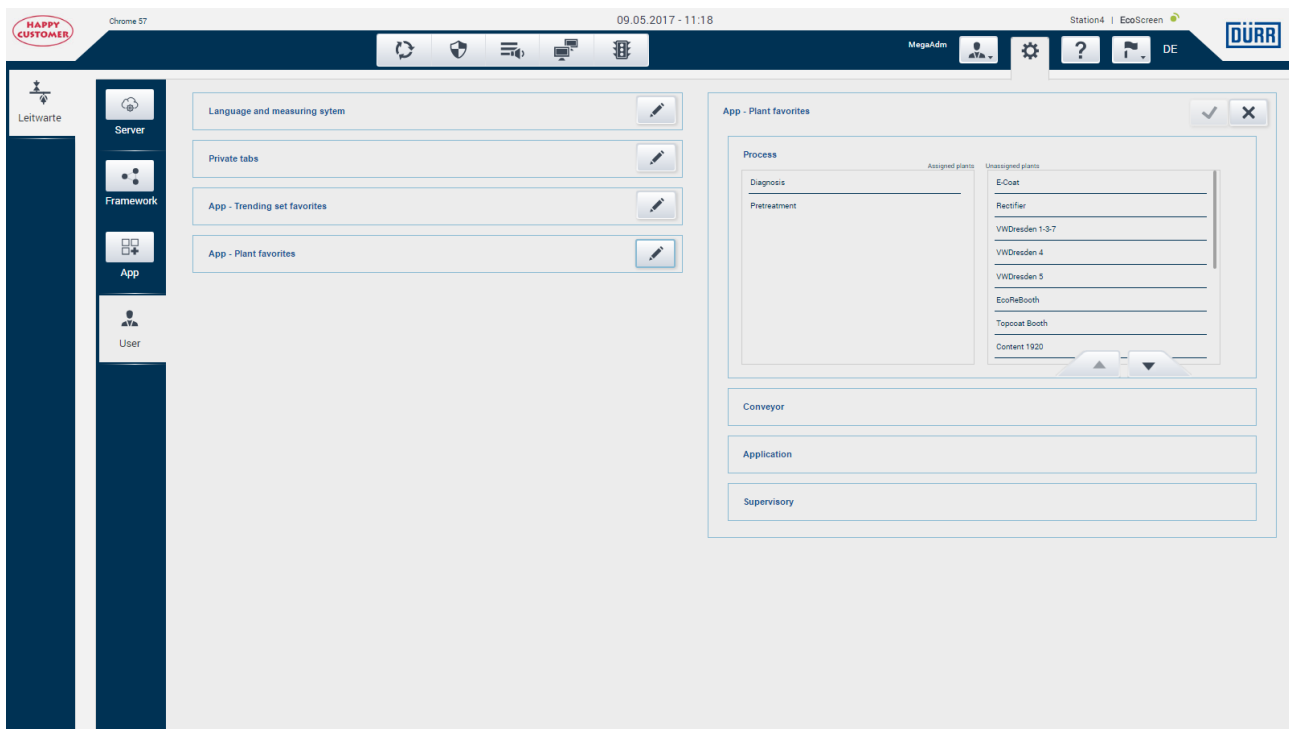


Unter dem Punkt „App – Plant favorites“ kann sich der User die Anlagen favorisieren, für die er im AlarmOverlay direkt die Reports haben möchte. Dafür wird eine Liste sämtlicher zur Abteilung gehörender Anlagen geladen und in „assigned/unassigned“ aufgeteilt.

[6.20.setPlantFavorites\(\)](#)

Durch Doppelklick in die rechte oder linke Spalte wird eine Anlage aus- oder abgewählt. Gespeichert wird das dann per

[6.21.savePlantFavorites\(\)](#)



3.4 Hilfe



Ein Klick auf das „?“ sorgt dafür, dass das Help-Overlay gebaut wird:

```
if ($(this).hasClass('active')) {
    $('.helpholder').remove();
    $(this).removeClass('active');
} else {
```

6.60.closeOther("#help"):

```
$(this).addClass('active');
```

```
if($('.helpholder').length === 0){
```

```
    $("<div/>", {
```

```
        "class": 'helpholder'
```

```
    }).appendTo('main');
```

```
    $("<div/>", {
```

```
        "class": 'boxHolder',
```

```
        "id": 'boxHolder'
```

```
    }).appendTo('.helpholder');
```

```
    $("<div/>", {
```

```
        "class": 'docBox',
```

```
        "id": "docBox",
```

```
        "html": "<p>Online Dokumentation</p>"
```

```
    }).appendTo('.boxHolder');
```

```
    $("<div/>", {
```

```
        "class": 'aboutBox',
```

```
        "id": 'aboutBox',
```

```
        "html": "<p>About Dürr HMI</p>"
```

```
    }).appendTo('.boxHolder');
```

```
emosWS.rest.framework.getFileInfo({
```

```
    path: serverRoot + 'config/app/' + myArr.app + '.json',
```

```
    success: function(info){
```

```
        var date = new Date(info.lastModified);
```

```
        $('.appDate').text(date.getFullYear() + '/' + (date.getMonth() + 1) + '/' +  
date.getDate() + ' ' + date.getHours() + ':' + date.getMinutes());
```

```
    },
```

```
    error: function(){
```

```
        console.log("request failed");
```

```
    },
```

```
    server: serverPool[0]
```

```
});
```

```
    $("<div/>", {
```

```

        "class": 'Groupbox',
        "html": "<span>App</span>"
    }).appendTo('.aboutBox').append('<span>'+ myArr.app
+'</span><br>').append('<span class="appDate"></span>');

    $("<div/>", {
        "class": 'Groupbox',
        "id": "serverlist",
        "html": "<span>Webserver</span>"
    }).appendTo('.aboutBox');
    var serverTable = $("<table/>", {
        "class": "serverTable"
    });
    $('#serverlist').append(serverTable);
    servers.forEach(function(server){
        var tmpTd = getId();
        $('<tr><td class="servername">'+ server.name +'</td><td id="'+ tmpTd + '"
class="servername warn"></td></tr>').appendTo('.serverTable');
        try {
            server.getStatus(function(status){
                if(status.reachable !== false){
                    $('#'+ tmpTd).removeClass('.warn').addClass('fine');
                }
            });
        } catch(e){}

    });

    $("<div/>", {
        "class": 'Groupbox',
        "html": "<span>Framework</span>"
    }).appendTo('.aboutBox').append('<span>0.9.5 Build
239</span><br>').append('<span>31.05.2017</span>');

    emosWS.rest.framework.getFileInfo({
        path: serverRoot + '/js/durr.emos.js',
        success: function(info){
            var date = new Date(info.lastModified);

```

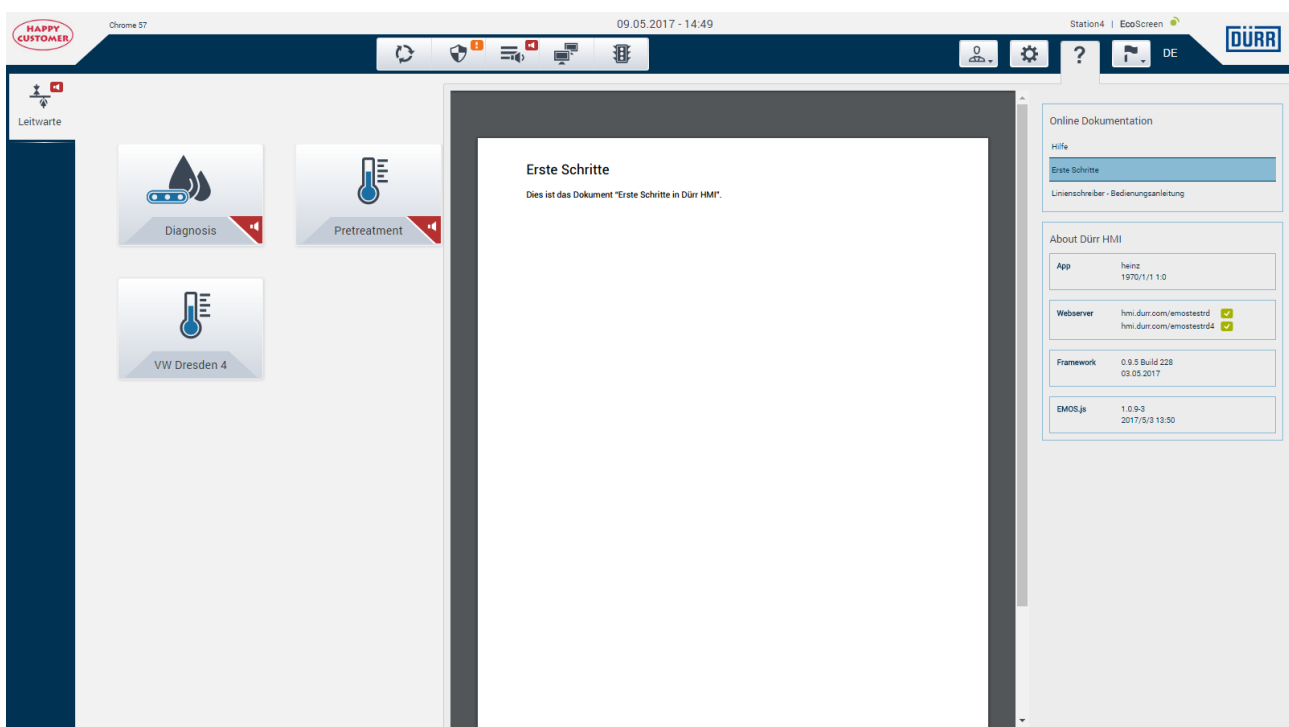
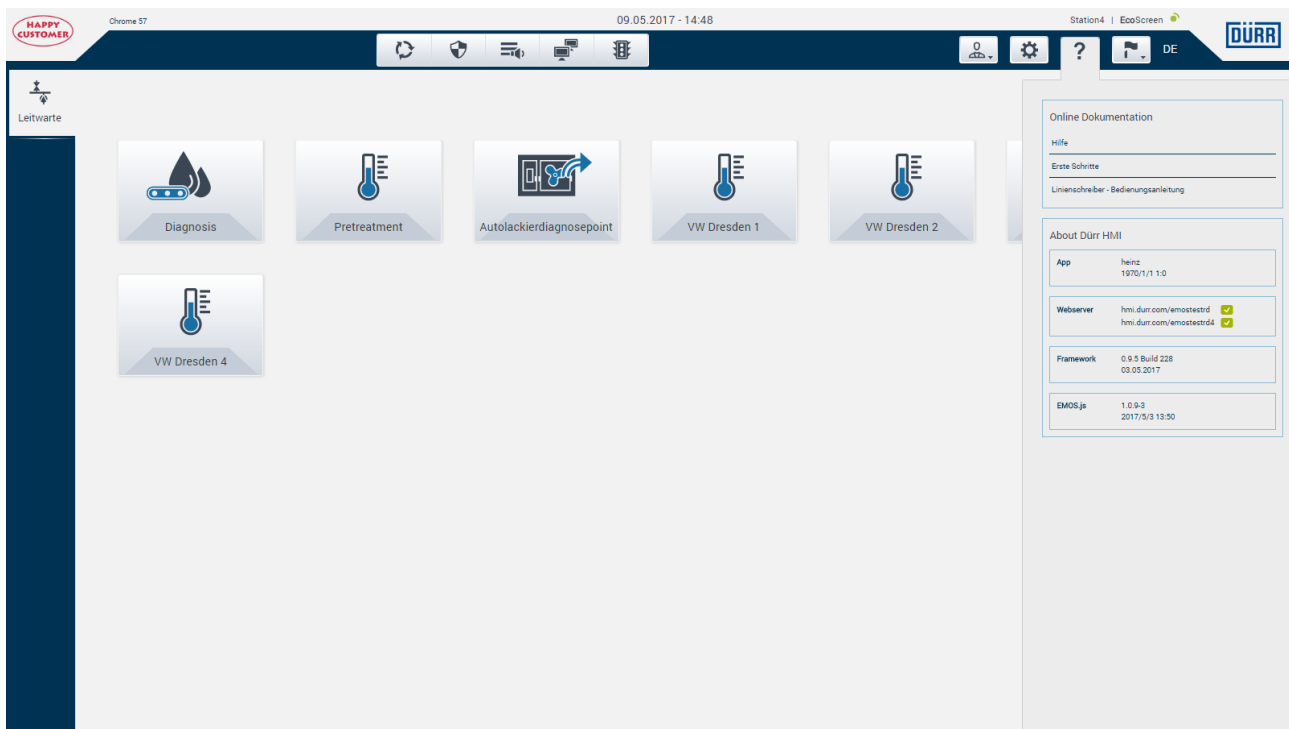
```

        $('.emosDate').text(date.getFullYear() + '/' + (date.getMonth() + 1) + '/' +
date.getDate() + ' ' + date.getHours() + ':' + date.getMinutes());
    },
    error: function(){
        console.log("request failed");
    },
    server: serverPool[0]
});
$("<div/>", {
    "class": 'Groupbox',
    "html": "<span>EMOS.js</span>"
}).appendTo('.aboutBox').append('<span>'+ emosWS.version
+'</span><br>').append('<span class="emosDate"></span>');
};
$('.helpholder').addClass('full');
6.58.getHelpObject\(\);
}

```

Hier werden nicht nur Hilfe-Dateien in der jeweiligen Sprache angeboten, es gibt auch nützliche Informationen zu:

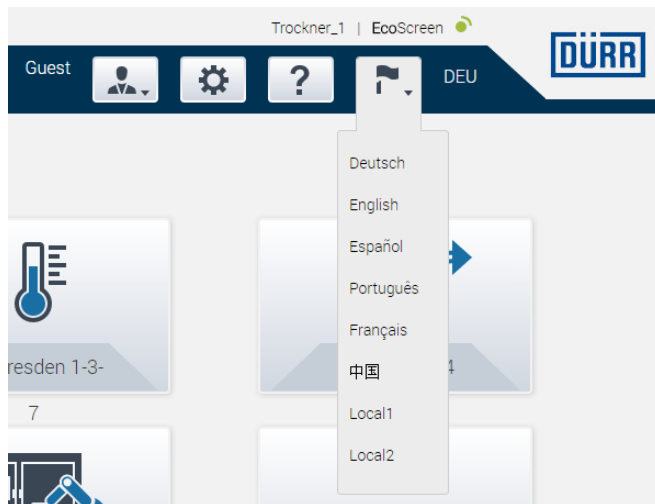
- App
- Webserver
- Framework
- Emos.js



3.5 Sprachauswahl



Die für die Applikation zur Verfügung stehenden Sprachen werden, wie bereits erwähnt, in der json-Datei hinterlegt. Klickt der User auf den Button Sprachauswahl, so öffnet sich ein Overlay mit einer Liste der Sprachen.



Ein Klick auf die gewünschte Sprache liest die hinterlegte Sprach-ID aus und sendet sie an das `emos.js`

```
emosWS.setLanguage(langId);
```

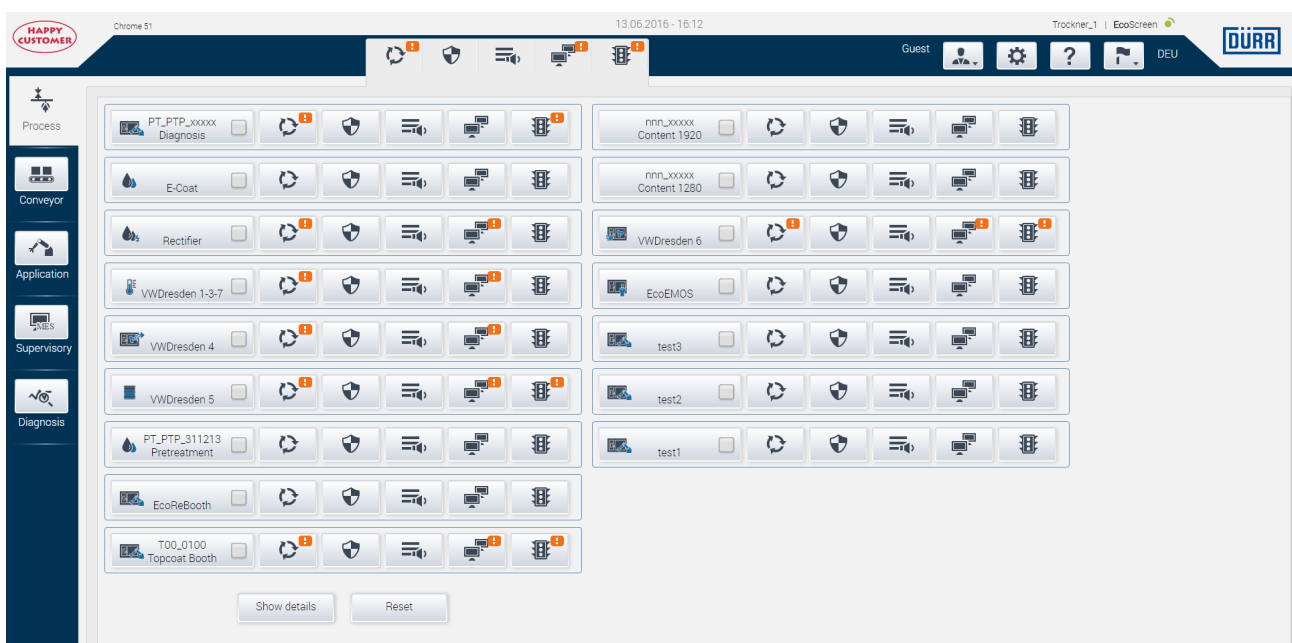
Durch die `emos`-Funktion `setAdviseText` wird dann jeder Text, der mit dieser Funktion verknüpft ist, automatisch in die soeben eingestellte Sprache übersetzt, sofern diese Übersetzung in der Datenbank vorliegt. z.B.

```
emosWS.sendAdviseText(val.textId, "name", function (msg) {
    text.innerHTML = msg.value;
});
```

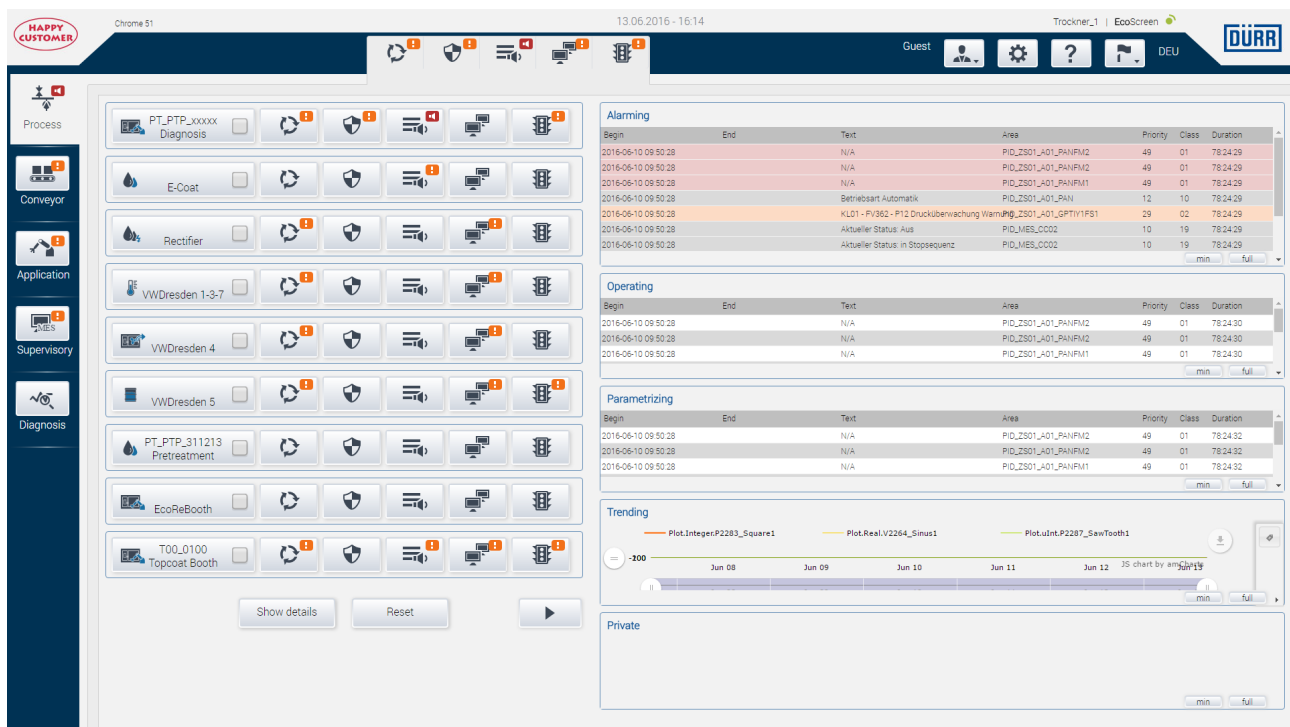
3.6 AlarmOverlay



Ein Klick auf die Alarmzeile öffnet das AlarmOverlay



Der Aufbau des Alarmoverlays wurde oben bereits beschrieben. Es soll als schnelle Informationsschicht, bei der man auch die Informationen mehrerer Anlage anzeigen lassen kann. Dazu wählt man die gewünschten Anlagen aus(Checkbox) und klickt danach auf Show details. Aber Vorsicht, wenn man nicht auf die Checkbox klickt, sondern auf den Button selbst, wird das entsprechende Anlagenbild(General_Plant_Overview) geladen.



Die Informationen für Alarmin, Operating, Parametrizing, Trending und Private werden nun auf der rechten Hälfte des Overlay angezeigt. Wurden die Anlagen auf dem Overlay mehrspaltig dargestellt, so ist nun nur noch eine Spalte sichtbar.

Zum Laden und Anzeigen dieser Informationen rufen wir die Funktion

[6.66.getDetailsOverlay\(\)](#)

die zum einen die Container und dazugehörigen Buttons erstellt zum anderen die gewünschten Daten von der emos.js anfordert. Z.B:

```
//mainbox
$(<div/>', {
    "class": "detailOverlay"
}).appendTo('#alarmpage_content').css("left", $('#column_0').width()
+ 10);

// Alarming-Box
$(<div/>', {
    "class": "alarmDetails detailAlarming norm",
    "id": "detailAlarming"
}).appendTo('.detailOverlay');

$(<div/>', {
    "class": "detailHead",
    "text": "Alarming"
}).appendTo('.detailAlarming');

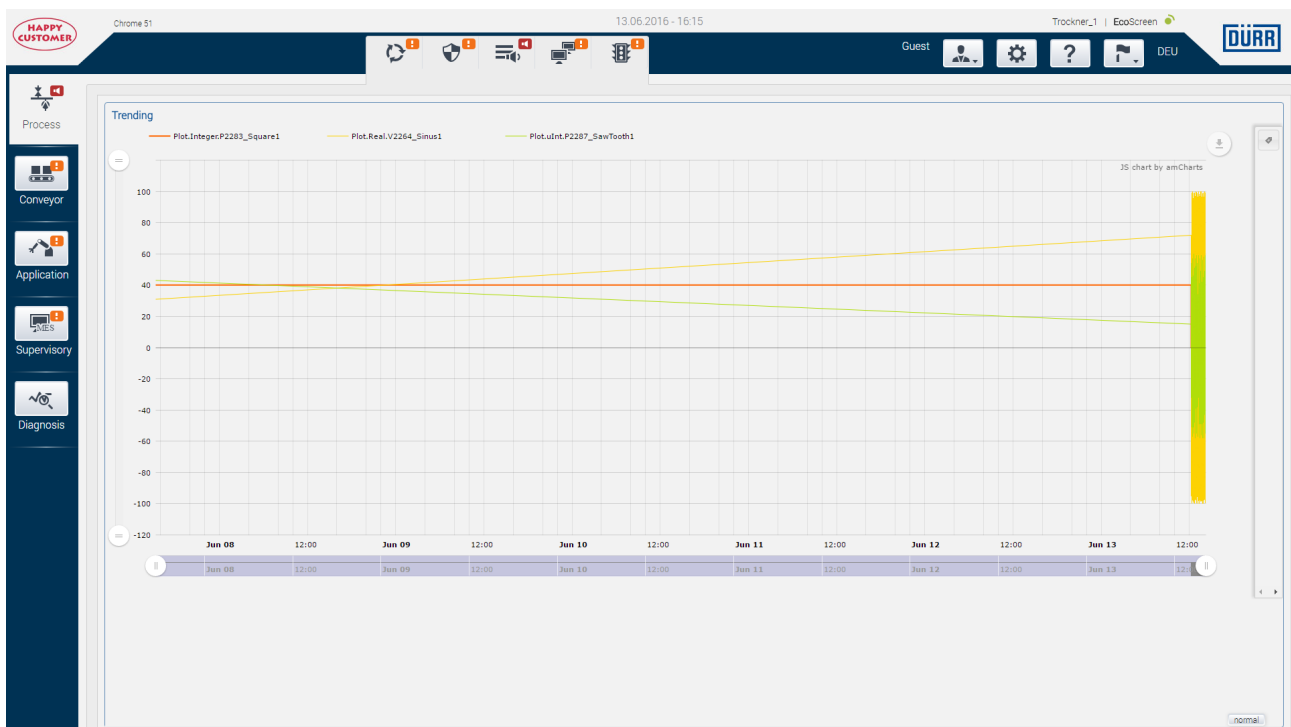
$(<div/>', {
```

```

        "class": "detailTable detailAlarmingTable"
    }).appendTo('.detailAlarming');
    $('.detailAlarming')[0].orgHeight = $('.detailAlarming').height();
    emosWS.rest.alarm.top('140_05V', 100, function (data) {
        var AlarmHistory = new emosWS.RestAlarmClient($
        ('.detailAlarmingTable'), data);
    });
    makeButtons('alarmingFooter', '.detailAlarming');

```

Um dennoch die anderen Spalten zu sehen, wurde hier eine Scrollfunktion implementiert. Die Informationsboxen selbst beinhalten 2 Buttons, einen zum verkleinern, einem um die Box in den Vollbildmodus zu bringen. Beim Verkleinern der Box wird der dadurch freigewordene Raum automatisch an die übrigen Boxen verteilt.



Die Buttons werden wie folgt aufgebaut:

```

function makeButtons(id, parent){
    $('<div/>', {
        "id": id,
        "class": "detailOverlayFooter"
    }).appendTo(parent);
    $('<span/>', {
        "class": "minimize emosbutton",
        "text": "min",
        "click": function(){
            if($(this).text() === 'min'){
                var freeHeight = $(this).closest('.alarmDetails').height() -

```

35;

```
$(this).closest('.alarmDetails').attr('style', '');

$(this).parent().css('background',
'none').parent().removeClass('norm').removeClass('bigger').addClass('mini');//.animate({'height': 'auto'})

$(this).parent().prev().find('table').hide();
$(this).text('normal');

var numberOfNormHeight = $('.detailOverlay
.alarmDetails.norm').size();
var heightPerBlock = freeHeight / numberOfNormHeight;
$.each($('.alarmDetails.norm'), function (key, val) {
    $(val).css({
        "height": $(val).height() + heightPerBlock,
        "max-height": $(val).height() + heightPerBlock
    }).addClass('bigger');
});
} else {
    var freeHeight = $
(this).closest('.alarmDetails').prop('orgHeight') - 35;
    var numberOfNormHeight = $('.detailOverlay
.alarmDetails.bigger').size();
    var heightPerBlock = freeHeight / numberOfNormHeight;
    $.each($('.alarmDetails.bigger'), function (key, val) {
        $(val).css({
            "height": $(val).height() - heightPerBlock,
            "max-height": $(val).height() - heightPerBlock
        })
    });
    $(this).parent().css('background',
'#f2f2f2').parent().removeClass('mini').addClass('norm');
    $(this).parent().prev().find('table').show();
    $(this).text('min');
}
}
}).appendTo('#' + id);
$('<span/>', {
    "class": "maximize emosbutton",
    "text": "full",
    "click": function(){
        if($(this).text() === 'full'){
            $(this).parent().prev().find('table').show();
            $('.detailOverlay').addClass('full');
            $('alarmDetails:not(#'+ $
(this).closest('.alarmDetails').attr('id') + ')').hide();
```

```

        $(this).text('normal').prev().hide();
    } else {
        $('.detailOverlay').removeClass('full');
        $('.alarmDetails').show();
        $(this).text('full').prev().show();
    }
}
}).appendTo('#' + id);
}

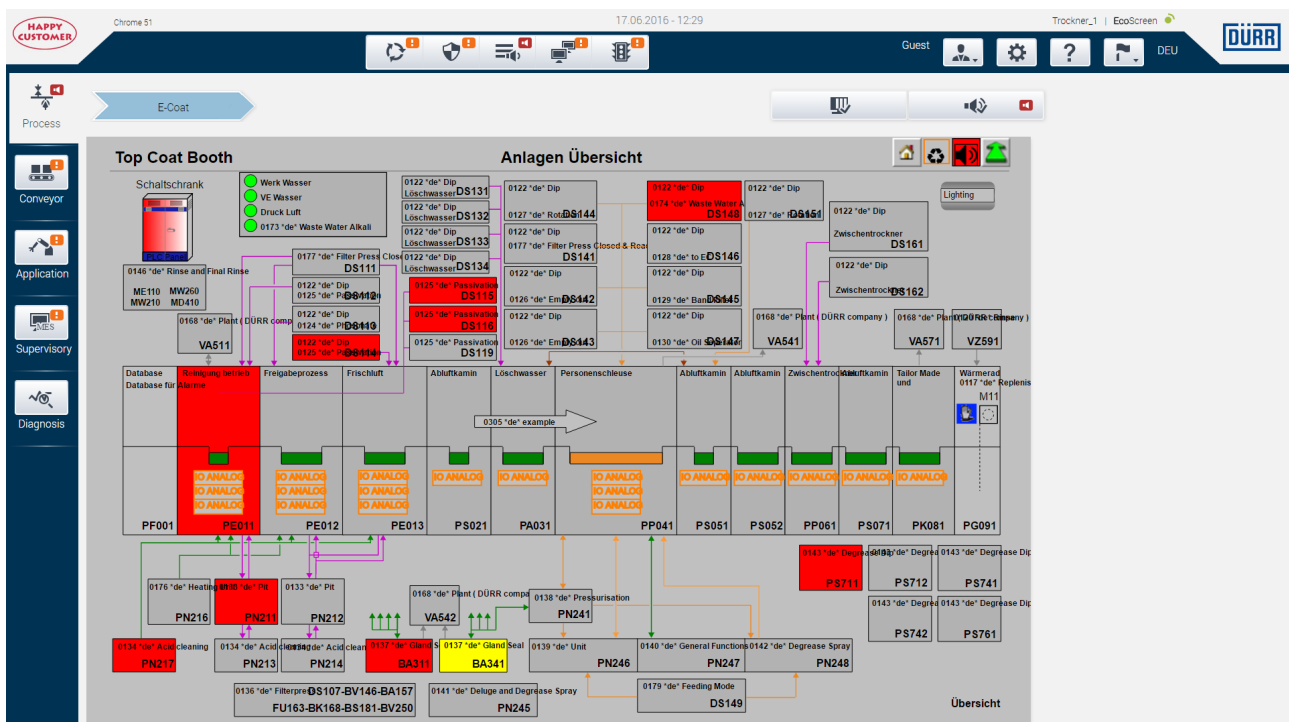
```

4 Main-Menü Funktionen

Die bisherigen Funktionen des Hauptmenüs wurden oben bereits beschrieben. Da der Aufbau der Menü-Funktionen der unterschiedlichen Abteilungen unterscheidet, werden diese nach Fertigstellung hier Dokumentiert.

5 Content-Bereich Funktionen

Nach Start des Dürr HMI Frameworks wird das Frontend mit den Daten aus der json-Datei aufgebaut(&app=diagnosis), als Beispielaufbau wird die Applikation „Process“ beschrieben. Die zu Process gehörenden Anlagen werden als Kachel im Content-Bereich angezeigt. Auf diesen Kacheln werden Alarm- und Warnungs-Icons angezeigt, sobald innerhalb der entsprechenden Anlage ein oder mehrere Alarme/Warnungen vorliegen. Möchte ein Mitarbeiter nun herausfinden, wo dieser Alarm/Warnung seinen Ursprung hat, klickt er auf diese Kachel. Nun wird das Übersichtsbild dieser Anlage geladen, eine sog. Breadcrumb-Navigation erstellt und Buttons zum Bestätigen am Alarmserver und Quittieren in der SPS.



Der Klick auf eine Kachel erstellt einen iFrame und lädt das zu der Anlage gehörende Übersichtsbild. Dazu werden noch 2 EventListener angelegt, OpenDiagnosisWindow und SiteProperties.

```
$('#center').empty();
$('#center').append('<div class="frameWrapper"></div>');
$('#<iframe id="plants" name="plants" src="' +
e.target.getAttribute('data-link') + '">').appendTo('.frameWrapper');
emosWS.addEventListener("OpenDiagnosisWindow",
onOpenDiagnosisWindow);
$('#plants').load(function (ev) {
    emosWS.addEventListener("SiteProperties", onSiteProperties);
    setTimeout(function () {
        clickedTile = $(e.target)[0].innerText;
        $('#breadcrumb li.bread_li:first-child').text(clickedTile);
    }, 200);
    var myMsg = {
        'type': "DiagnosisWindowMode",
        'mode': 1
    };
    var myElem = document.getElementById('plants').contentWindow;
    myElem.postMessage(JSON.stringify(myMsg), '*');

});
```

der EventListener SiteProperties liefert uns ein Objekt mit folgenden Informationen:

```
AckALM:Object
BreadcrumbLevel:0
Links:Array[28]
QuitALM:Object
Trends:Array[3]
```

Mit den Infos zu AckALM und QuitALM werden die Buttons



bestückt, um die aktuellen Alarme zu bestätigen bzw. zu quittieren. Durch die Information zum BreadcrumbLevel und Links wird die Breadcrumb-Navigation gebaut. Der BreadcrumbLevel gibt an, auf welcher Hierarchieebene sich das aktuelle Bild befindet und Links enthält sämtliche Links zu den Bildern, die vom aktuellen Bild erreichbar sind. Diese Link-Liste wird bei einem Mouseover über den Breadcrumbeintrag angezeigt. Klick auf einen dieser Links lädt dann das entsprechende Bild. Natürlich wird dann auch wieder die Breadcrumb-Navigation und die entspr. Linklisten generiert.

besteht auch die Möglichkeit, mehrere Informationsfenster zu öffnen, vom gleichen Anlagenabschnitt, aber auch von unterschiedlichen Anlagenabschnitten um diese vergleichen zu können. Jeder eingeloggte Mitarbeiter hat zudem die Möglichkeit, sich individuelle Tabs zusammenzustellen, sog. Private-Tabs, denen sämtliche gewünschten Informationen (Betrieb/Meldungen/Parameter) angeordnet werden können.

Jeder einzelne Tab ist zum einen über mehrere Seiten, als auch über mehr Inhalt verfügen, als auf dem Bildschirm angezeigt werden kann. Somit existiert eine vertikale und horizontale Scrollfunktion, die direkt anzeigt, wenn es etwas zum Scrollen gibt. Liegen mehrere Seiten eines Tabs vor, wird zusätzlich noch ein Pager eingeblendet, der zeigt, um wie viele Seiten es sich handelt und welche aktuell angezeigt wird. Auch verfügt der Pager über ein Quickselekt-Menü, welches eine Schnellauswahl ermöglicht.

5.1 Quick Alarm-Buttons

Wie oben bereits beschrieben, erhalten wir über die Site-Properties die Informationen, die wir für die beiden Buttons benötigen, um die aktuellen Alarmer zu bestätigen bzw. zu quittieren. Wir erstellen die Buttons und die benötigte Funktionalität zum Bestätigen bzw. Quittieren:

```
if (msg.QuitALM) {
    $("<span/>", {
        "class": "emosbutton reset",
        "id": "reset"
    }).appendTo('#quickcontrol');
    $('#reset')[0].myPlc = msg.QuitALM.PlcTag;
}

if (msg.AckALM) {
    $("<span/>", {
        "class": "emosbutton quit",
        "id": "quit"
    }).appendTo('#quickcontrol');
    $("<span/>", {
        "class": "",
        "id": "quit_alarm"
    }).appendTo('#quit');
    new emosWS.HTMLFaultWarning({
        "id": 'quit_alarm',
        "alarmGroup": msg.AckALM.AlarmGroups
    });
    $('#quit')[0].myPlc = msg.AckALM.AlarmGroups;
}
}
```

```

}
$('#quickcontrol').on("click", "#quit", function () {
    $.each($('#quit')[0].myPlc, function (key, val) {
        emosWS.poke(val + '.Alarm', 1);
    });
});
$('#quickcontrol').on("click", "#reset", function () {
    emosWS.poke($('#reset')[0].myPlc + '.Alarm', 1);
});

```

5.2 BreadCrumb-Navigation

Das Objekt, welches wir über die Site-Properties erhalten, gibt uns die Information, auf welcher Hierarchieebene das aktuelle Bild liegt und welche Links auf ihm platziert sind.

```

if (msg) {
    var iframelink = $('#plants').contents().get(0).URL;
    var filename = iframelink.substring(iframelink.lastIndexOf('/') +
1);

    var idname = filename.slice(0, -4);
    filename = (filename.slice(0, -4)).replace(/\/_/g, " ");
    if (msg.BreadcrumbLevel === breadparts.length - 1) {
        var delLinklist = filenames[msg.BreadcrumbLevel].replace(/\/ /g,
'_');

        breadparts.splice(msg.BreadcrumbLevel);
        breadurls.splice(msg.BreadcrumbLevel);
        filenames.splice(msg.BreadcrumbLevel);
        delete linklist[delLinklist];
    }
    if (jQuery.inArray(filename, filenames) === -1) {
        filenames.push(filename);
        breadurls.push(iframelink);
        breadparts.push("<li id='" + idname + "' class='bread_li' data-
link='" + filename + "'>" + filename + "</li>");
    }
    $('#breadcrumb').empty();
    $('#quickcontrol').empty();
    ulMaker('bread', breadparts, '#breadcrumb');
    $('#breadcrumb li:first-child').text('');
    linklist[idname] = [];
    $.each(msg.Links, function (key, val) {
        linklist[idname].push("<li class='breadoverlay_li' data-link='"
+ val + "'>" + val + "</li>");
    });
    var i = 0;

```

```

    for (var key in linklist) {
        if (linklist[key].length > 0) {
            ulMaker('breadoverlay' + i, linklist[key], '#breadcrumb');
            $('#breadoverlay' + i).addClass(key + ' crumbOL');
        }
        i++;
    }
}

```

In einer BreadCrumb-Navigation ist natürlich auch möglich, wieder auf das „Mutterbild“ zu gelangen. Klickt der Mitarbeiter also z.B. auf „E-Coat“ soll die General_Plant_Overview.html in den iFrame geladen werden und die Navigation muss dementsprechend angepasst werden.

```

$('#breadcrumb').click(function (e) {

    $('.diagnose').remove();
    clearMyInterval();
    terminateActiveAlarmClient();
    var myText = e.target.getAttribute('data-link');
    var breadtext = jQuery.inArray(myText, filenames);
    if (breadtext !== -1) {
        var delLinklist = filenames[breadtext + 1].replace(/\\ /g, '_');
        breadparts.splice(breadtext + 1);
        breadurls.splice(breadtext + 1);
        filenames.splice(breadtext + 1);
        delete linklist[delLinklist];
        $('#plants').attr('src', breadurls[breadurls.length - 1]);
    }
});

```

5.3 OpenDiagnosisWindow

Da das „alte“ Design und die Funktionen der Informationsfenster aus der emos.js kommen und da bei einigen Testszenarien die benötigten Daten nicht vom Server gesendet werden, sondern von der emos selbst übergeben werden, wurde auch das neue Design dort integriert. Es könnte sich später als sinnvoll erweisen, dass, wenn bekannt ist, wie die anderen Abteilungen die an dieser Stelle benötigten Daten ausliefern, der Aufbau des Informationsfensters aus der emos ausgegliedert werden sollte.

Bei klick auf ein Element wird vom Server ein Datenobjekt gesendet, welches die Informationen enthält, mit welchen Elementen und Werten das Statusfenster bestückt werden muss. Ein kleiner Auszug aus einem solchen Objekt:

```

windowData:Object
  Comment:""
  Head:Object
  Name:"FU245.01"
  OPCVersionID:"GP.E2003_Vari"

```

```

PageControl:Object
    1000004-AddPageTab:Object
    1000007-AddText:Object
    1000008-AddTextLED:Object
        Border:"B"
        OPCID:"SB.S1251_Auto"
        OffLED:"O"
        OnLED:"G"
        Position:"L"
        Text:"Automatic Mode"
        TextID:"S1251"
        VisibleOPCID:""
        writable:false
    1000009-AddTextLED:Object
    1000010-AddText:Object
    1000011-AddTextInputLED:Object
    1000012-AddTextInputLED:Object
    1000013-AddTextInputLED:Object
    1000014-AddTextInputLED:Object
    1000015-AddText:Object
    1000016-AddTextLED:Object
    1000017-AddTextLED:Object
    1000018-AddTextLED:Object
    1000019-AddTextLED:Object
    1000006-Messages:Object
    1000020-AddPageTab:Object
    1999990-AddPageTab:Object
    1999993-AddPageTab:Object
    Text:"Operation"
    TextID:"Operation"

```

Es gibt im Grunde 3 Darstellungen des Informationsfensters: das eigentliche Fenster, die Kopie eines Fensters und ein Vergleichsfenster. In der Funktion onMessage, in der das Datenobjekt erhalten wird, wird zunächst abgeklärt, welche Darstellungsart benötigt und ob das Maximum an darstellbaren Fenstern bereits erreicht ist

```

var counter = $(".diagnose").get().length;
    if (counter === 4 || (window.screen.width < 1900 && counter === 3)) {
        return;
    }
msg.myCounter = myCounter;
if (compareWin) {
    msg.additionalClass = 'fixedPart';
    anotherWin = true;
}

```

```

    }
    if ($('#.diagnose') && anotherWin !== true) {
        $('#.diagnose').remove();
        clearMyInterval();
        terminateActiveAlarmClient();
    }
}

```

Das jeweils erste Fenster ist am rechten Rand fixiert, Fensterkopien oder Vergleichsfenster jeweils rechts vom vorherigen Fenster angeordnet.

5.3.1 Das Statusfenster

Damit das Fenster aufgebaut werden kann, wird das Datenobjekt an die `emos.js` gesendet

```
var tmp = new emosWS.HTMLDiagnosisWindow(msg);
```

Die Klasse `HTMLDiagnosisWindow` übernimmt dann. Hier wird das HTML-Gerüst gebaut und anschließend mit Daten gefüllt. Für die unterschiedlichen Elemente, die Information oder Einstellmöglichkeiten bieten, liegen jeweils eigene Klassen vor. Das Datenobjekt enthält die Informationen, welche Klassen in welcher Reihenfolge benötigt werden. Dafür durchläuft dann das Objekt eine Schleife mit einer `switch-case`-Anweisung um je nach `ObjectType` die benötigte Elementklasse zu holen:

```

switch (objectType) {
    case "AlarmClient":
        new AlarmClient(parent, elementData, this.PLC +
this.ID, extra);
        break;
    case "AddText":
        new DiagnosisText(parent, elementData, this.PLC +
this.ID, this);
        break;
    case "AddString":
        new DiagnosisString(parent, elementData, this.PLC +
this.ID, false, this, key);
        break;
    case "AddStringInput":
        new DiagnosisString(parent, elementData, this.PLC +
this.ID, true, this, key);
        break;
}

```

Die Elementklasse liefert dann den Aufbau und die Daten zurück. Mit den obigen Beispieldaten würde der Aufbau des Operation-Tabs wie folgt ablaufen:

1000004-AddPageTab: erstellt ein neues Tab mit der Headline „Operation“

1000007-AddText: erstellt eine „Parentbox“ mit der Headline „Operation Modes“ innerhalb des Operation-Tabs

1000008-AddTextLED: erstellt eine LED-Anzeige mit dem Text „Automatic Mode“

innerhalb der Parentbox Operation Modes

1000009-AddTextLED: erstellt eine LED-Anzeige mit dem Text „Manual Mode“
innerhalb der Parentbox Operation Modes

1000010-AddText: erstellt eine „Parentbox“ mit der Headline „Control Signals“
innerhalb des Operation-Tabs

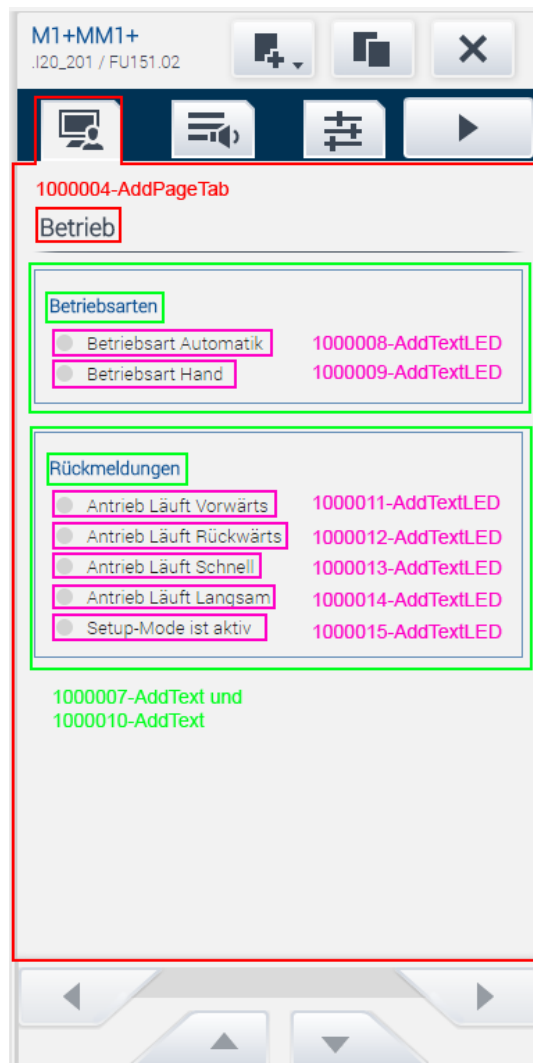
1000011-AddTextInputLED: erstellt eine LED-Anzeige mit dem Text „Drive Forward“
innerhalb der Parentbox Control Signals

1000012-AddTextInputLED: erstellt eine LED-Anzeige mit dem Text „Drive Reverse“
innerhalb der Parentbox Control Signals

1000013-AddTextInputLED: erstellt eine LED-Anzeige mit dem Text „Drive Fast“
innerhalb der Parentbox Control Signals

1000014-AddTextInputLED: erstellt eine LED-Anzeige mit dem Text „Drive Slow“
innerhalb der Parentbox Control Signals

1000015-AddTextInputLED: erstellt eine LED-Anzeige mit dem Text „Setup Mode
Active“ innerhalb der Parentbox Control Signals



Der grundsätzliche Aufbau der Elementklassen ist allen gleich. Es wird das Html-Gerüst erstellt und mit Daten befüllt. Die Textdaten, sofern vorhanden, werden per `emosWS.sendAdviseText` überwacht, damit sie in der richtigen Sprache angezeigt werden und sich auch bei Sprachauswahl anpassen. Auch andere Daten, die sich während der Laufzeit ändern können, werden überwacht,

```
emosWS.advise(this.plctagLed, this.datachangeLedListener.bind(this), "",
emosWS.tagType.IO);
```

Dies kann eine Meldung sein, eine Änderung der Betriebsart, eine Umstellung von Soll-/Istzeiten etc.

Ist das Element beschreibbar? Was soll auf Klick passieren? OK, bei den Klick-Funktionen gibt es Unterschiede: vorausgesetzt der Mitarbeiter ist eingeloggt und hat die benötigten Rechte um Änderungen durchzuführen, gibt es für ihn Toggle-Buttons für Ein-/Ausschalter, Zahlenfelder, Textfelder, Zeit- und Datumsfelder. Die änderbaren Felder sind mit einem write-Button versehen. Zum Bearbeiten der Zahlenfelder wird ein Keypad eingeblendet, welches per jQuery widget gebaut wurde (`$.widget("ems.numpad")`). In dem Widget wird die Html-Struktur aufgebaut und Klick- und Mouseup/down Funktionen gesetzt.

Bei den Textfeldern wird eine virtuelle Tastatur eingeblendet. Diese wurde nicht selbst gebaut, sonder durch ein externes Script eingebunden, da dieses Script eine sehr große Auswahl an verschiedenen Sprachen bietet. Zum jetzigen Zeitpunkt sind aber noch nicht alle benötigten Sprachen zugeordnet. Dazu muss aber nur die Sprach-ID im keyboard.js eingegeben werden. Z.B. ist dort deutsch mit der ID 70 eingetragen. Da es einige solcher Anpassungen in der keyboard.js und keyboard.css gibt, sollten diese Scripte nicht geupdatet werden, da es ansonsten einige Fehlfunktionen geben könnte.

Für den Date-Picker wurde mobiscroll eingesetzt, da dieses Plugin bereits das „Look and Feel“ eines Smartphone-Datepickers mitbringt.

5.3.2 Fensterduplikate/ Vergleichsfenster



Im Head des Informationsfensters befinden sich 3 Buttons:

- Duplizieren und öffnen eines Kindfensters
- Vergleichsfenster
- Schließen

Das Duplizieren eines Informationsfenster gleicht dem öffnen eines solchen, nur dass das Duplikat eine zusätzliche CSS-Klasse und eine Nummer benötigt. Es muss errechnet werden, wie viele Informationsfenster bereits geöffnet sind und an welche Position das neue gesetzt werden muss.

```
$('#center').on("click", ".arrangeTabs", function (e) {
    var msg = {};
    var newMsg = [];
```

```

var counter = $(".diagnose").get().length;
if (counter === 4 || (window.screen.width < 1900 && counter === 3)) {
    return;
}
myCounter++;
newMsg = $(this).closest('.diagnose').find('.DiagnoseBody')[0].msg;
newMsg.additionalClass = 'fixedPart';
newMsg.myCounter = myCounter;
onMessage(newMsg, true);
var actWidth = $(this).closest('.diagnose').width();
var actPos = $(this).closest('.diagnose').css('right');
setTimeout(function () {
    $('.diagnose.' + myCounter).css('right', actPos);
    $('.diagnose.' + myCounter).animate({
        'right': (actWidth * counter)
    });
    $('.diagnose.' + myCounter).find('.DiagnosisFooter').animate({
        'right': (actWidth * counter)
    });
}, 200);
});

```

Verfügt das Informationsfenster über ein- oder mehrere Kindfenster, so erscheint ein Popdownmenü unterhalb des Duplizierenbuttons sobald dieser mit der Maus überfahren wird. Dort kann dann gewählt werden, ob eine Kopie des aktuellen Fensters geöffnet werden soll, oder eines der angezeigten Kindfenster.

Der Button für das Vergleichsfenster sorgt zuerst dafür, dass sich sämtliche zusätzlich geöffneten Informationsfenster per Animation auf die rechte Seite zusammenklappen. Dann wird eine Meldung gezeigt, dass man sich ein anderen Prozess aussuchen möchte. Durch das setzen der Variable `compareWin = true;` werden vorhandene Informationsfenster nicht gelöscht, es wird aber das neue gewünschte geöffnet und dann werden alle Fenster per Animation in ihre richtigen Positionen gesetzt. Natürlich können mehrere Vergleichsfenster und oder Duplikate geöffnet werden, die maximale Anzahl an geöffneten Fenstern ist auf 4, bei Full-HD-Auflösung und auf 3 bei kleineren Bildschirmen begrenzt.

Ist die maximal Anzahl an Fenstern erreicht, werden beide Buttons deaktiviert.

Der Schließen-Button entfernt das gewünschte Informationsfenster, positioniert ggf. nachfolgende Fenster neu und beendet nicht mehr benötigte Dienste

```

clearMyInterval(this);
terminateActiveAlarmClient();

```


5.3.3 Scrollbars

Sobald das Informationsfenster aufgebaut wurde, oder ein anderer Tab sichtbar wird, überprüfen wir zuerst, ob dieser Tab vertikal scrollbar sein muss

```
function showScroll(pageElement) {
    $.each($('.DiagnosisPageBlank:visible'), function () {
        var page;
        if ($(pageElement).hasClass('DiagnosisPageBlank')) {
            page = $(this);
        } else {
            page = $(this).find('.DiagnosisSubPageBlank:visible');
        }
        var extraheight = 25;
        if ($('#html').hasClass('Firefox')) {
            extraheight = 25;
        }

        var contentelement = $(this).closest('.DiagnosisContent');
        var bottomParent = page.height(); //page.position().top +
        var elemTop = page.scrollTop();
        var elemHight = page.prop("scrollHeight") - extraheight;
        if (elemTop === 0 && elemHight > bottomParent + 1) {
            contentelement.next().find('.scrollUp').removeClass('active');
            contentelement.next().find('.scrollDown').addClass('active');
        } else if (elemTop === 0 && elemHight <= bottomParent + 1) {
            contentelement.next().find('.scrollUp').removeClass('active');
            contentelement.next().find('.scrollDown').removeClass('active');
        } else if (elemTop > 0 && elemHight - elemTop <= bottomParent + 1) {
            contentelement.next().find('.scrollDown').removeClass('active');
            contentelement.next().find('.scrollUp').addClass('active');
        } else {
            contentelement.next().find('.scrollUp').addClass('active');
            contentelement.next().find('.scrollDown').addClass('active');
        }
    });
}
```

Diese Funktion wird im Übrigen immer aufgerufen, sobald sich der Tab oder etwas innerhalb des Tabs ändert(wie z.B. das öffnen/schließen der Textfelder oder auch nach einem scrollen.). Muss der Tab nun scrollbar sein, es befindet sich also mehr Content in ihm als angezeigt werden kann, so wird der entsprechende Button als aktiv gekennzeichnet. Klickt man dann zum scrollen auf einen der scrollButtons, wird die scrollUpDown Funktion ausgeführt.

```

function scrollUpDown(myTarget, updown){
    var myMainPage =
myTarget.parent().prev().find('.DiagnosisPageBlank:visible');
    var contentElement = '';
    if (myMainPage.find('.DiagnosisSubPageBlank:visible').length > 0) {
        contentElement = myMainPage.find('.DiagnosisSubPageBlank:visible');
    } else {
        contentElement = myMainPage;
    }
    var myNum = (updown === 'up' ) ? + 42 : - 42;
    var scrollnumber = contentElement.height() + myNum;
    var scrollToNum = (updown === 'up' ) ? - scrollnumber : + scrollnumber;
    contentElement.animate({
        scrollTop: contentElement.scrollTop() + scrollToNum
    }, 1000, function () {
        showScroll($(contentElement));
    });
}

```

Als nächstes wird überprüft, ob der aktuelle Tab über Subtabs verfügt. Wenn dem so ist, wird neben der Tab-Headline die Funktion für das horizontale Scrollen und ein Pager benötigt. Diesen Aufbau übernimmt die buildHeader-Funktion.

```

function buildHeader(pageId) {
    var waytofooter = $(
(pageId).closest('.DiagnosisContent').next('.DiagnosisFooter');
    var waytofooterhead = waytofooter.children('.footerHead');
    waytofooterhead.children('.circle').remove();
    waytofooterhead.removeClass('sub');
    waytofooter.children('.scrollLeft').removeClass('active');
    waytofooter.children('.scrollRight').removeClass('active');
    waytofooter.find('#subTabMenu').remove();
    var pagerHeadlineId = $(
(pageId).prevAll('.pager').children('.pagerHead').children('.pagerHeadLine')).attr('id');
    var pagerHeadlineTextId;
    var PagerHeadlinetext;
    if ($(pageId).find('.DiagnosisSubPageBlank:visible').length === 0) {
        pagerHeadlineTextId = $(pageId)[0].headId;
        PagerHeadlinetext = $(pageId)[0].head;
    } else {
        var menulines = [];
        pagerHeadlineTextId = $(
(pageId).children().find('.DiagnosisSubPageBlank:visible')[0].headId;

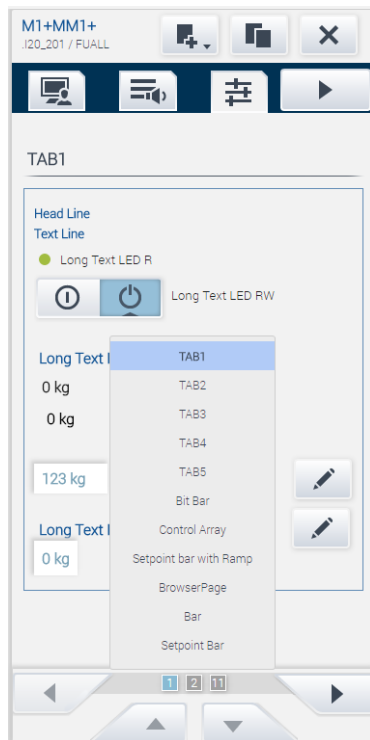
```

```

        PagerHeadlinetext = $(
(pageId).children().find('.DiagnosisSubPageBlank:visible')[0].head;
        for (var i = 0; i < $(pageId).children('.diasubtabs')[0].tabNumbers;
i++) {
            waytofooterhead.append('<span class="circle c' + i + '"><span
class="circlenum">' + (i + 1) + '</span></span>');
            menulines.push("<li class='subtab' data-obj='" + i + "'">" + $(
(pageId).children().find('.DiagnosisSubPageBlank')[i].head + "</li>");
        }
        ulMaker('subTabMenu', menulines, waytofooterhead);
        waytofooterhead.addClass('sub');
        waytofooterhead.children('.circle').eq($
(pageId).find('.DiagnosisSubPageBlank:visible')[0].tabNum - 1).addClass('big');
        waytofooterhead.children('.subTabMenu').children('li').eq($
(pageId).find('.DiagnosisSubPageBlank:visible')[0].tabNum -
1).addClass('active');
        if ($(pageId).find('.DiagnosisSubPageBlank:visible')[0].tabNum < $
(pageId).children('.diasubtabs')[0].tabNumbers) {
            waytofooter.children('.scrollRight').addClass('active');
        }
        if ($(pageId).find('.DiagnosisSubPageBlank:visible')[0].tabNum > 1)
{
            waytofooter.children('.scrollLeft').addClass('active');
        }
    }
    var text = document.getElementById(pagerHeadlineId);
    var tmpAdviceText = emosWS.sendAdviseText(pagerHeadlineTextId, "name",
function (msg) {
        if (msg) {
            text.value = msg.value;
        }
    });
    if (!tmpAdviceText) {
        text.value = PagerHeadlinetext;
    }
    if ($(pageId).hasClass('Private')) {
        $(text).prop('disabled', false);
    } else {
        $(text).prop('disabled', true);
    }
}

```

Möchte nun ein Mitarbeiter durch die Subtabs scrollen, kann er dies nun mittels der rechts/links Scrollbuttons oder durch eine Seitenauswahl im Pager, welcher sichtbar wird, wenn mit der Maus über den Bereich zwischen den rechts/links Scrollbuttons gefahren wird.



Ein Klick im Pager oder auf einen der rechts/links Buttons löst im Endeffekt die Funktion `scrollLeftRight` aus,

```
function scrollLeftRight(direction, nonDirection, nextVisibleElement,
nextElement, newPosition, self) {
    $(self).closest('.DiagnosisFooter').find('.circle').removeClass('big');
    $
    (self).closest('.DiagnosisFooter').find('.subTabMenu').children('li').removeClass('active');
    nextElement.show();
    nextElement.animate({
        'right': 0
    });
    nextVisibleElement.animate({
        'right': newPosition
    }, function () {
        nextVisibleElement.scrollTop(0);
        nextVisibleElement.hide();
        showScroll(nextElement);
        $(self).closest('.DiagnosisFooter').children('.scroll' +
nonDirection).addClass('active');
    });
    if (direction === 'Right' && nextElement.next().length === 0 ||
direction === 'Left' && nextElement.prev().length === 0) {
        $(self).closest('.DiagnosisFooter').children('.scroll' +
direction).removeClass('active');
    }
}
```

```

        var pagerHeadlineId = $(self).closest('.DiagnosisFooter').prev().find('.pager').children('.pagerHead').children('.pagerHeadLine').attr('id');

        var text = document.getElementById(pagerHeadlineId);

        var tmpAdviceText = emosWS.sendAdviseText(nextElement[0].headId, "name",
function (msg) {
            if (msg) {
                text.value = msg.value;
            }
        });
        if (!tmpAdviceText) {
            text.value = nextElement[0].head;
        }

        $(self).closest('.DiagnosisFooter').find('.circle').eq(nextElement[0].tabNum - 1).addClass('big');

        $(self).closest('.DiagnosisFooter').find('.subTabMenu').children('li').eq(nextElement[0].tabNum - 1).addClass('active');

        if ($('.circle').last().hasClass('big')) {
            $('.circle.big').prev().addClass('prevbig');
        } else {
            $('.circle').removeClass('prevbig');
        }
    }
}

```

welche bewirkt, dass der gewünschte Subtab per Animation sichtbar, die benötigte Headline gesetzt und im Pager die aktuelle Seite gekennzeichnet wird.

5.3.4 Private-Tabs

Jeder Tab enthält ein oder mehrere Elemente. Da ein Tab auch einige Subtabs enthalten kann und so die Anzahl an Elementen entsprechend groß wird, besteht die Möglichkeit für eingeloggte Mitarbeiter, eigene Tabs anzulegen, damit die für ihn wichtigen Elemente zusammen gefasst werden können. Dies kann er, in dem er auf die gewünschten Elemente mit einem Rechtsklick das Kontextmenü hervorruft. In diesem kann er dann auswählen, ob das Element in einen bestehenden Private-Tab oder in einen neuen kopiert werden soll.

Beim Aufbau des Informationsfensters wird bereits ein vorerst leerer Private-Tab durch hinzufügen entsprechender Daten in das oben bereits beschriebene Datenobjekt angelegt.

```

this.data['1999990-AddPageTab'] = {
    "1999991-AddPageControl": {
        "1999992-AddPageTab": {
            "Text": "Privat1",
            "TextID": "Privat1"
        }
    }
}

```

```

    }
  },
  "Text": "Private",
  "TextID": "Private"
};

```

Liegt von diesem Mitarbeiter bereits eine json-Datei mit seiner individuellen Private-Tab Belegung vor, so werden diese Daten dann ausgelesen, hat er noch keine Privates angelegt, werden statt dessen die vom Administrator angelegten Public-Tabs geladen, z.B.

```

"Private2": {
  "5": {
    "data": {
      "OnLED": "G",
      "OPCID": "SB.S2268_GrtF",
      "TextID": "S2268",
      "OffLED": "O",
      "Text": "Actual Value > Positive Fault Limit",
      "Bold": "1",
      "Border": "B",
      "VisibleOPCID": "SB.S3110_CBAPV",
      "VisibleAtRight": "0",
      "Position": "Long",
      "writable": false,
      "DiagnosisLibID": 1
    },
    "myPLC": "PT_PTP_311213.EF01_PE011_T35",
    "myKey": "1000008-AddTextLEDLong",
    "writable": false
  }
}

```

und der individuelle Private-Tab mit seinen Subtabs angelegt und sämtliche Elemente hinzugefügt:

```

function getPrivateTabs(preparent) {
  var privateResults = $.getPrivValues(); // laden der json-Datei
  if (privateResults) {
    $.each(Object.keys(privateResults), function (key, val) {
      var mother = preparent.find("[data-pos='" + val + "']");
      if (key === 0 && mother.length === 0) {
        mother = preparent.find(".dPage");
      }
      if (mother.length === 1) {
        mother[0].head = val;
        $(mother).attr("data-pos", val);
        $('#contextMenu .context a.first').text('Copy to Tab ' +
val);
      }
      var parent = preparent.find("[data-pos='" + val +
''']).children('table');

```

```

        if (parent.length === 0) {
            var tmp = new
emosWS.AddSubTab(preparent.children('.diasubtabs'), val);
            parent = preparent.find("[data-pos='" + val +
            "'").children('table');
        }
        $.each(Object.keys(privateResults[val]), function (key1, val1) {
            new emosWS.AddDataToSubTab(parent, privateResults[val]
            [val1].data, privateResults[val][val1].myPLC, privateResults[val][val1].myKey,
            privateResults[val][val1].writable, val1);
        });
    });
}
}

```

Durch `emosWS.AddSubTab` wird bei Bedarf ein neuer Subtab angelegt und dessen Name in das Kontextmenü übernommen.

Durch `emosWS.AddDataToSubTab` werden die Daten, je nach ihrem Datentyp in den aktuellen Subtab eingefügt. Die Klasse `AddDataToSub` dient hierbei nun als Schnittstelle um mit den eingangs erwähnten Elementklassen zu kommunizieren und die Daten bereitzustellen. Da die Elemente auf den Private-Tabs genauso arbeiten müssen, wie die Elemente in den Standard-Tabs, können wir also nicht einfach Kopien der Elemente machen, da sich die Überwachungsfunktionen auf die Element-ID's beziehen. So sind sie genauso funktional wie ihre Originale.

Möchte der Mitarbeiter nun weitere Elemente zu seinen Private-Tabs hinzufügen, werden beim öffnen des Kontextmenüs die benötigten Elementdaten ausgelesen und in Variablen festgehalten:

```

if (userloggedin) {
    if ($(this).hasClass('priv')) {
        $('.elementdelete a').show();
        $('.context').hide();
    } else {
        $('.context').show();
        $('.elementdelete a').hide();
    }
    e.preventDefault();
    x = e.clientX;
    y = e.clientY;
    $("#contextMenu").css("left", x + "px");
    $("#contextMenu").css("top", y + "px");
    $("#contextMenu").show();
    $(this).css('border', '1px solid #000');
    actElement = $(this);
    o = $(this)[0].myData;
    myKey = $(this)[0].myKey;
}

```

```

        myPrivId = $(this)[0].myPrivId;
        myPLC = $(this).closest('.DiagnoseBody')[0].msg.PLC + $
(this).closest('.DiagnoseBody')[0].msg.Tag;
        o.DiagnosisLibID = $(this).closest('.DiagnoseBody')
[0].msg.DiagnosisLibID;

```

Bein Auswahl des gewünschten Tabs werden diese dann zur Erstellung des neuen Elements und dessen Platzierung weitergegeben

```

$('body').on("click", ".subContext:not(.elementdelete)", function (e) {
    var pageId = $(e.target).attr('href');
    if (pageId === '#') {
        var tmp = new emosWS.AddSubTab($('dPage.Private .diasubtabs'));
        var parentelementid = tmp.parent[0].id;
        pageId += $(''#' + parentelementid).children().last().attr('id');
    }
    var parent = $(pageId).find('table');
    var writable = false;
    var objType = DiagnosisCommon.getObjectType(myKey);
    switch (objType) {
        case "AddStringInput":
        case "AddStringInputLong":
        case "AddTextDoubleTimerInputLongMin":
            o.writable = true;
            break;
        case "AddTextDoubleTimerLong":
            o.valueInSecond = true;
            break;
        case "AddTextDoubleTimerInputLong":
            o.writable = true;
            o.valueInSecond = true;
            break;
        case "AddTextIntegerInput":
        case "AddTextRealInput":
        case "AddTextRealInputNiveau":
        case "AddTextIntegerInputLong":
        case "AddTextRealInputLong":
        case "AddLEDTextIntegerInputLong":
        case "AddTextLEDIntegerInputLong":
        case "AddLEDTextRealInputLong":
        case "AddTextIntegerIntegerInputLong":
        case "AddTextRealRealInputLong":
            o.writable = true;
            break;
    }

```



```

        case "AddTextTimerInputLong":
        case "AddTextSiemensDateInputLong":
        case "AddTextSiemensTimeInputLong":
        case "AddTextSiemens_TIME_TimerInputLong":
        case "AddTextSiemens_S5TIME_TimerInputLong":
        case "AddTextSiemens_DATE_AND_TIME_TimerInputLong":
            o.writable = true;
            break;
    }
    var actId = 1;
    var myArray = [];
    var privateResults = $.getPrivValues();

    if (!privateResults) {
        privateResults = {};
        actId = 1;
    } else {
        privateResults = $.getPrivValues();
        $.each(Object.keys(privateResults), function (key, val) {
            $.each(Object.keys(privateResults[val]), function (key1, val1) {
                myArray.push(parseInt(val1));
            });
        });
        actId = Math.max.apply(Math, myArray);
        if (actId === -Infinity) {
            actId = 0;
        }
        actId++;
    }

    new emosWS.AddDataToSubTab(parent, o, myPLC, myKey, writable, actId);
    $('#contextMenu').hide();
    actElement.css('border', 'none');
    actElement = null;
    var tabNum = $(pageId).attr('data-pos');
    if (!privateResults[tabNum]) {
        privateResults[tabNum] = {};
    }
    privateResults[tabNum][actId] = {
        "data": o,
        "myPLC": myPLC,
        "myKey": myKey,
        "writable": writable
    }

```

```

    };

    if (myName) {
        emosWS.writeData("/framework/config/users/" + myName + ".json",
JSON.stringify(privateResults, null, " "));
    }
    o = null;
});

```

und das neue Element mit seinen Daten in das json geschrieben.

Natürlich kann ein hinzugefügtes Element auch wieder gelöscht werden. Befindet sich der Mitarbeiter auf seinem Private-Tab und ruft auf dem zu entfernenden Element erneut das Kontextmenü mit Hilfe der rechten Maustaste auf, so ist dort nun der Eintrag „delete element from Tab“. Wird nun auf diesen Eintrag geklickt, so wird das Element aus dem Tab und aus der json-Datei gelöscht:

```

var parent = actElement.closest('.dPage').attr("data-pos");
var privateResults = $.getPrivValues();
delete privateResults[parent][myPrivId];
var tmp = privateResults[parent];
if (jQuery.isEmptyObject(tmp)) {
    console.log('leer');
    //delete privateResults[parent];
}

emosWS.writeData("/framework/config/users/" + myName + ".json",
JSON.stringify(privateResults, null, " "));
actElement.remove();
$('#contextMenu').hide();

```

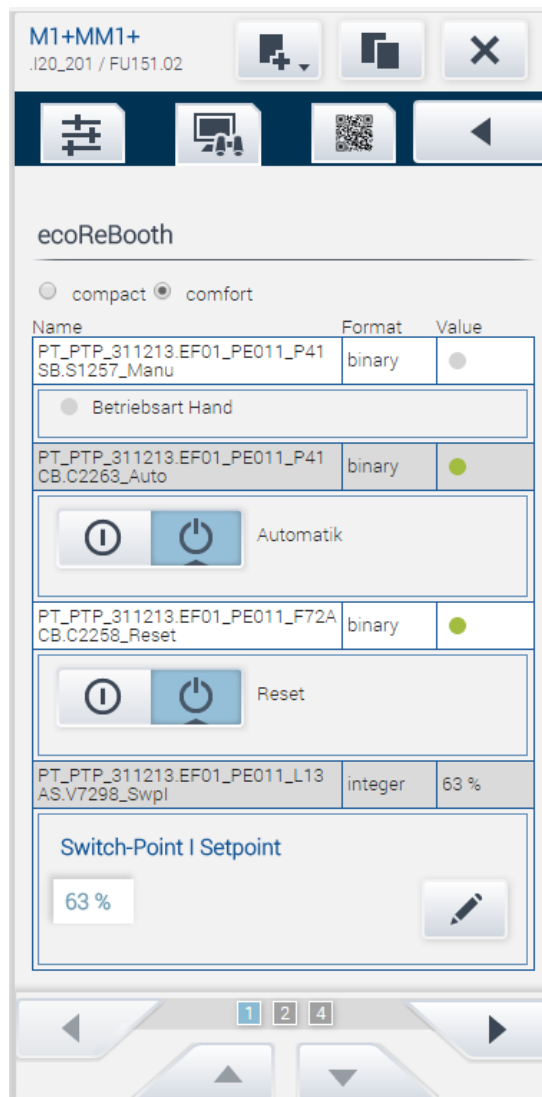
In der Grundausstattung werden die Private-Tabs mit „Private 1“ - „Private n“ benannt. Dies kann aber auch individuell eingestellt werden. Durch klick auf diese Headline wird das Textfeld aktiv und ein virtuelles Keyboard erscheint. Nach Eingabe eines neuen Namens wird dies nun auch in der json-Datei gespeichert:

```

var parent = $(self).closest('.diatabs
').find('.DiagnosisSubPageBlank:visible');
var oldAttr = parent.attr("data-pos");
parent.attr("data-pos", self.value);
parent[0].head = self.value;
var privateResults = JSON.stringify($.getPrivValues());
privateResults = privateResults.replace(new RegExp("\\b" + oldAttr +
"\\b"), self.value);
emosWS.writeData("/framework/config/users/" + myName + ".json",
privateResults);
$('#contextMenu .context a:contains(' + oldAttr + ')').text('Copy to Tab
' + self.value);

```

Es gibt für die Element auf den Private-Tabs zwei Darstellungen: „compact“ und „comfort“. Grundsätzlich werden die Elemente in der „Compact-Darstellung“ angezeigt, d.h. es werden nur Name, Format und Value in einer Tabelle gezeigt. Die Comfort-Darstellung zeigt das Element so, wie es im Haupt-Tab auch angezeigt wird. Der Mitarbeiter kann die Comfort-Darstellung für ein oder für sämtliche Elemente wählen. Um nur ein Element umzuschalten reicht ein Klick auf das gewünschte Element. Um sämtliche Elemente zu vergrößern, kann dies mit den Radio-Buttons oberhalb der Tabelle geschehen.



5.3.5 Weitere Funktionen folgen

6 Funktionen

Hier folgt nur eine Auflistung sämtlicher Funktionen, die in der „functions.js“ zu finden sind, mit einer kurzen Beschreibung.

6.1 *getTimeCodes(parent)*

Damit in den Alarmreports die vom angemeldeten User gewünschte lokale Zeit angezeigt

werden kann, benötigen wir den LanguageCode. Diesen kann der angemeldete User in den Einstellungen unter dem Punkt USER → Language and Measuring auswählen und abspeichern. Auch kann er an dieser Stelle angeben, ob er die Zeit im 12h oder 24h Modus sehen möchte. Diese Funktion lädt eine Liste mit sämtlichen LanguageCodes und erstellt ein Listenelement aus jedem Eintrag. Dabei wird auch direkt geprüft, ob der User bereits eine Local Time hat. Es dann entweder, bei vorhandener localTime seine, oder bei nicht vorhandener, die allgemeine LocalTime als aktiv gekennzeichnet. Danach wird noch die Checkbox 12h/24h geprüft und ggf. gesetzt.

6.2 GetServerList()

Diese Funktion baut für jeden Server, der in der globalen Variable „servers“ vorhanden ist, eine eigene Box mit den Button „A“, „delete“ und „edit“. Wenn ein User angemeldet ist und Administratorrechte hat, sind diese Buttons klickbar, ansonsten disabled. Die Server werden auch direkt auf ihren Status überprüft. Der Status wird mittels eines grünen bzw. orangen Fähnchens angezeigt. Es werden auch die Boxen für den/die CAS-Server und den Chatserver aufgebaut. Zusätzlich gibt es noch den Button „new Server“, um einen weiteren Server anzulegen.

6.3 NewServer()

Diese Funktion erstellt den Kopf für eine Serverseite, damit der Administrator einen neuen Server anlegen kann. Hier wird zuerst die Möglichkeit geschaffen, einen Servernamen anzugeben. Zum Aufbau dieses Textfeldes wird die Funktion

[6.34.buildInputGroup\('Host name', 'Host name', 'Host name', ''\)](#)

genutzt.

6.4 newServerBody()

Diese Funktion wird aufgerufen, um einige Textfelder zu generieren. Diese werden benötigt, damit z.B. Webservices, die auf dem Server installiert sind, auch den anderen Servern zur Verfügung gestellt werden können. Auch die Verbindung zum CAS-Server wird hier angegeben.

6.5 saveServerName(dataFile)

dataFile = Inhalt des Textfeldes Servername / ("input[name='Host name']").val()

Wenn der „save-Button“ geklickt wird, wird diese Funktion aufgerufen, um den neuen bzw. geänderten Server abzuspeichern. Dazu wird der Servername in den serverPool(globale Variable) eingetragen. Der serverPool beinhaltet sämtliche Server, die in der site.configurations.js auf dem aktuellen Server eingetragen sind. Für jeden vorhandenen Server wird sodann dessen site.configuration.js geladen, diese dann soweit zerpfückt(tempData.split('\n')), dass die Zeile „serverPool“ auf das Vorhandensein des neuen Servers geprüft und ggf. ergänzt werden kann. Dann wird diese veränderte Datei wieder zurückgeschrieben und der neue Server steht den bekannten Servern zur Verfügung.

6.6 saveServer(dataFile, self)

dataFile = Inhalt des Textfeldes Servername / ("input[name='Host name']").val()

self = der geklickte save-Button

Sind die Elemente aus newServerBody() vorhanden, so sorgt der Klick auf den save-Button dafür, dass die Informationen ausgelesen werden(welcher Webservice steht welchem Server zur Verfügung) und sendet diese Informationen dann mittels emos-Funktion

emosWS.rest.configuration.updateConfiguration
um die Serverkonfiguration zu updaten.

6.7 deleteServerName(dataFile)

dataFile = Attribut data-name des geklickten delete-Buttons

Klickt der Administrator auf den „delete-Button“ wird diese Funktion ausgeführt. Hier wird nun auch von jedem bekannten Server(serverPool) die site.configurations.js geladen, gesplittet und der zu löschende Server entfernt. Dann wird die geänderte Datei wieder abgespeichert.

6.8 editServer(dataFile, editserver, parent)

dataFile = Attribut data-name des geklickten edit-Buttons

editserver = wenn true dann editieren eines Servers, false bedeutet neuer Server

parent = Parent des edit-Buttons

Würde ein neuer Servername gespeichert oder soll ein bestehender Server bearbeitet werden, wird diese Funktion aufgerufen. Diese sorgt zuerst dafür, dass sämtliche zur Verfügung stehende Daten angezeigt werden. Im Falle des Bearbeitens eines vorhandenen Servers wird dessen Name durch den Aufruf der

[6.3.NewServer\(\)](#) Funktion

geladen und die vorhandenen Webservices sowie die URL zum CAS-Server angezeigt. Ein neuer Server erhält diese Felder ohne Inhalte. Dazu wird die Funktion

[6.4.newServerBody\(\)](#)

aufgerufen um die dann erhaltenen Textfelder mit ihrem Inhalt, soweit vorhanden, zu füllen. Um die Inhalt zu bekommen, wird die

emosWS.rest.configuration.getConfiguraton Funktion
aufgerufen.

6.9 showAdminpage(dataFile, parent)

dataFile = Attribut data-name des geklickten A-Buttons

parent = \$('#servercenter')

Klickt der User auf den „A-Button“ in einer Serverbox, kommt diese Funktion zum Einsatz. Sie sorgt dafür, dass ein Container angelegt wird, in den dann mittels iFrame die Serverinformationen der Administrationsseite geladen wird.

6.10 *editCentralCasServer(parent)*

parent = parent des geklickten edit-Buttons

Möchte der Administrator den CentralCasServer editieren, wird nach Klick auf den edit-Button diese Funktion ausgeführt. Es wird zuerst per

emosWS.rest.configuration.getConfiguration

die allgemeine Konfiguration geladen und bei Erfolg eine Box für den CentralCas aufgebaut, die 2 Textfelder enthält, die dann die Url für das centralCAS.FrontEnd bzw centralCAS.BackEnd beinhalten.

6.11 *getLanguageList()*

Diese Funktion, die beim starten der App ausgeführt wird, verarbeitet die aus der fw_globals.json geladenen Sprachen zu Checkboxes und einem LanguageDropDown.

6.12 *saveLanguageList()*

Hierdurch werden die vom Administrator per Checkbox ausgewählten Sprachen gespeichert. Dazu wird in der fw_globals.json zu der jeweiligen Sprache bei „selected“ ein „checked“ oder „unchecked“ geschrieben. Dann wird noch die Funktion

[6.22.buildLanguageDropDown\(\)](#)

ausgeführt.

6.13 *saveLogo()*

Damit ein ausgewähltes Logo-Image auf den Server geladen wird, nimmt diese Funktion den Namen der Datei und übergibt ihn der emos-Funktion

emosWS.rest.framework.uploadFile.

Die Datei wird dabei an die URL aus der site.configurations.js, uploadImage, gesendet und erhält auch den dortigen Namen(ist im Moment zumindest noch so, damit die CSS-Anweisungen noch greifen).

6.14 *saveSettings()*

Hat der Administrator eine default Sprache und ein default Einheitensystem gewählt und möchte dies speichern, kommt diese Funktion, nimmt die gewünschten Daten und speichert diese in der fw_globals.json. Hat dieser User selbst keine Sprache und kein Einheitensystem für sich ausgewählt, so werden diese dann auch direkt gesetzt. Nach dem Speichern wird der save-Button wieder deaktiviert.

6.15 *saveOptions()*

Auch diese Funktion nimmt die entsprechenden Daten und speichert diese in der fw_globals.json.

6.16 *saveUserLanguage()*

Hat ein eingeloggter User eine Sprache, Einheitensystem, LanguageCode und 12h/24 Einstellung gewählt, werden diese zuerst in das privateResults Objekt geschrieben. Dann werden die Sprache und das Einheitensystem direkt gesetzt

```
emosWS.setLanguage($('.configcenterRight .languagemeasuring  
.language').find('li.active').attr('data-value'));
```

```
emosWS.setUnit($('.configcenterRight .languagemeasuring .measuring').attr('data-  
value'));
```

und die Daten in die username.json geschrieben.

```
emosWS.writeData(writeJSONURL + myName + ".json", JSON.stringify(privateResults,  
null, " "));
```

6.17 *getprivateTabList()*

Generiert eine Liste sämtlicher PrivateTabs mit click-Funktion um den Eintrag als aktiv/deaktiv zu kennzeichnen.

6.18 *getTrendingSetList()*

Um eine Liste der vorhandenen Trendingsets zu erhalten, wird zuerst die

```
emosWS.trendFileREST.getInfo
```

ausgeführt. Mit dem erhaltenen Ergebnis wird dann in einer Schleife jedes Set identifiziert und zu einer Liste hinzugefügt, welches von der aktuell eingeloggten Person stammt. Auch hier wird die click-Funktion aktiv/deaktiv hinzugefügt. Die Informationen über das Set wird noch dem DOM-Objekt hinzugefügt. Damit der User sich für die unterschiedlichen Hauptbereiche(Process, Conveyor, Supervisory, Application) auch unterschiedliche Favoriten setzen kann, wird die dann erstellte Liste geklont und dem entsprechenden Bereich übergeben. Hat der User bereits Favoriten gesetzt, werden die entsprechenden Einträge als active gekennzeichnet → dann sichtbar durch einen blauen Hintergrund.

Da diese Listen dann mehr Platz einnehmen, als der Screen zur Verfügung stellt, werden noch Scroll-Buttons implementiert

```
6.62.scrollUpDownButton($('.configcenterRight', '.preferences.full', '.Groupbox', 'scrollUp');
```

```
6.62.scrollUpDownButton($('.configcenterRight', '.preferences.full', '.Groupbox',  
'scrollDown');
```

und die

```
6.56.showScroll($('.configcenterRight'))
```

aufgerufen.

6.19 *saveTrendingSetFavorites()*

Für jeden der Hauptbereiche wird nach einem aktiven Eintrag gesucht. Ist dieser vorhanden, werden die Daten aus dem DOM-Objekt genommen und einem Zwischenobjekt übergeben. Sind alle aktiven gefunden, wird dieses Zwischenobjekt an privateResults.trendingsets übergeben und mittels

```
emosWS.writeData(writeJSONURL + myName + ".json", JSON.stringify(privateResults,  
null, " "));
```

gespeichert. Anschliessend wird der save-Button wieder deaktiviert.

6.20 ***setPlantFavorites()***

Diese Funktion arbeitet noch nicht mit den richtigen Daten, es wird nur eine vorgefertigte Liste mit einigen Plants geladen.

Für jeden Hauptbereich wird eine Liste möglicher Anlagen vom Server gesendet und diese dann in den entsprechenden Boxen aufgelistet. Für jeden Eintrag in diesen Listen wird dann ein Listenelement erstellt, mit einer doppelklick-Funktion. Beim Erstellen wird noch geprüft, ob diese Anlage bereits ein Favorit ist. Wenn dem so ist, wird diese Anlage in die assigned-Box ansonsten in die unassigned-Box geschrieben. Die doppelklick-Funktion sorgt dafür, dass ein Element von einer Box in die andere transportiert wird. Da auch hier mehr Daten als Platz vorhanden sind, werden noch Scroll-Buttons

[6.62.scrollUpDownButton](#)('unassignedBox', '.preferences.full', '.Groupbox', 'scrollUp');

[6.62.scrollUpDownButton](#)('unassignedBox', '.preferences.full', '.Groupbox', 'scrollDown');

hinzugefügt und die

[6.56.showScroll](#)(\$('.unassignedBox'))

aufgerufen.

6.21 ***savePlantFavorites()***

Aus der assignedBox werden sämtliche Listeneinträge gelesen und deren Text in ein Array geschrieben. Dieses Array wird dann an privateResults.plants übergeben und mittels

`emosWS.writeData(writeJSONURL + myName + ".json", JSON.stringify(privateResults, null, " "));`

gespeichert. Der save-Button wird dann wieder deaktiviert.

6.22 ***buildLanguageDropDown()***

Für jeden Eintrag in languageList wird ein Element erzeugt. Zusätzlich noch ein Trenner und die Listenelemente für „metrisch“ und „imperial“. Daraus wird dann noch eine unsortierte Liste erstellt. Stellt das DropDown „Language“ im Header dar.

6.23 ***getFolder(path)***

path = z.b. '../config/app'

Liest den Inhalt eines Ordners auf dem Server aus.

6.24 ***buildAppList(parent)***

Erstellt aus der gelieferten Liste

[6.23.getFolder](#)('../config/app')

für jede App eine Box. Da es mehr Boxen sein können als Platz vorhanden ist, rufen wir die

[6.56.showScroll](#)(\$('.configcenter.full'))

auf, um ggf. scrollen zu können.

6.25 ***buildClassList(path, id, selector)***

path = z.b. '../config/app'

id = eine ID halt

selector = ist der Eintrag im DropDown, der als 'active' gekennzeichnet werden soll

Mit Hilfe des übergebenen Pfades, wird der entsprechende Ordner ausgelesen

[6.23.getFolder\(path\)](#)

Mit den erhaltenen Daten werden dann ein DropDown-Menü erstellt, bzw. dessen Listenelemente.

6.26 *buildButtonList(buttonList, parent, addClass, sidebuttonNum)*

buttonList = Liste der Elemente aus denen die Button-Liste generiert werden soll
(Sidebuttons, Plants etc)

parent = das Element, die sämtliche Buttons aufnehmen soll

addClass = additional class, z.b. 'sidebutton'

sidebuttonNum = Nummer des Sidebuttons (Nummerierung durch Reihenfolge in json)

Erstellt die Button-Boxen für die Sidebuttons, Homestations, Plants und Alarmbuttons

6.27 *newApp(editapp)*

editapp = wenn true, soll eine App editiert werden, ansonsten eine neue angelegt.

Die Funktion stellt im Grunde nur die Rahmen für eine Applikation her. Textfeld für den Namen der App wird per

[6.34.buildInputGroup\('New App', 'App name', 'App name', ''\)](#)

Funktion gebildet, der Rahmen für die Sidebuttons und den Rahmen für die Station homes als HTML-Gerüst generiert.

6.28 *addSidebutton(addNew, selector, datanum)*

addNew = wenn true, soll ein Sidebutton editiert werden, ansonsten ein neuer angelegt.

selector = ist der Eintrag im DropDown, der als 'active' gekennzeichnet werden soll

datanum = Nummerierung des Sidebuttons

Die addSidebutton Funktion liefert das Template für die Sidebutton Informationen. Es wird sowohl bei einem neuen Button als auch zum editieren eines Button aufgerufen. Soll ein neuer Button erzeugt werden (addNew), wird zuerst ein Objekt für die Daten angelegt. Das Template erhält 3 Textfelder

[6.34.buildInputGroup\('New Sidebutton', 'Sidebutton name', 'name', ''\)](#)

[6.34.buildInputGroup\('New Text ID', 'Text ID', 'textId', 'fl'\)](#)

[6.34.buildInputGroup\('New Alarm group', 'Alarm group', 'alarm', 'fl'\)](#)

und ein PullDown, aus dem die Iconklasse ausgewählt werden kann

[6.25.buildClassList\('svg/Mainmenu', id, selector\)](#)

6.29 *addPlant(addNew, sidebuttonNum, selector, datanum)*

addNew = wenn true, soll eine Anlage editiert werden, ansonsten eine neue angelegt.

sidebuttonNum = Nummer des Sidebuttons, für die Zuordnung

selector = ist der Eintrag im DropDown, der als 'active' gekennzeichnet werden soll

datanum = Nummerierung der Anlage

Ähnlich wie die addSidebutton liefert die addPlant das Template um die Informationen für eine Anlage aufzunehmen. Wird auch beim editieren oder neu anlegen einer Anlage benötigt. Nur gibt es hier 7 Textfelder

[6.34.buildInputGroup](#)('New Plant', 'Plant name', 'name', 'fl')

[6.34.buildInputGroup](#)('New Security Group','Security group', 'alarm_security', '')

[6.34.buildInputGroup](#)('New Text ID', 'Text ID', 'textId', 'fl')

[6.34.buildInputGroup](#)('New Alarm group', 'Alarm group', 'alarm', '')

[6.34.buildInputGroup](#)('New Content link', 'Link', 'link', 'cl full')

[6.34.buildInputGroup](#)('New PLC topic name', 'PLC topic name(CPU)', 'plc', 'fl')

[6.34.buildInputGroup](#)('New Central Cabinet name', 'Central Cabinet name', 'cabinetname', 'fl')

und 2 Pulldowns

[6.25.buildClassList](#)('svg/icons_start', id, selector)

[6.25.buildClassList](#)('../config/plcsoftware', id2,
appResults.sidebuttons[sidebuttonNum].tiles[datanum].plcsoftware.replace(/_/g, " "))

6.30 *addAlarmbutton(addNew, sidebuttonNum, selector, datanum)*

addNew = wenn true, soll ein Alarmbutton editiert werden, ansonsten ein neuer angelegt.

sidebuttonNum = Nummer des Sidebuttons, für die Zuordnung

selector = ist der Eintrag im DropDown, der als 'active' gekennzeichnet werden soll

datanum = Nummerierung des Alarmbuttons

Erstellt das Alarmbutton-Template

6.31 *addHome(addNew, datanum)*

addNew = wenn true, soll ein Homebutton editiert werden, ansonsten ein neuer angelegt.

datanum = Nummerierung des Homebuttons

Erstellt das HomeButton Template

6.32 *getCancel(e, server)*

e = Element auf das geklickt wurde

server = true / false

Schliesst eine geöffnete Applikation und erstellt die Appliste neu

6.33 *hideElement(e)*

e = Element auf das geklickt wurde

Sozusagen ein canceln der Einagbe.

6.34 *buildInputGroup(valueText, textLabel, textName, setFloat)*

valueText = Placeholder im Textfeld

textLabel = Label Text

textName = name des input-Feldes

setFloat = ", 'fl'(float:left), 'full'(langes inputFeld für URL), 'cl full'(clear float, dann Full-Size)

Erstellt einen Block dessen Hauptteil ein inputfield ist.

6.35 *editApp(parent, dataFile, duplicateFile)*

parent = Elternelement des geklickten edit-Buttons

dataFile = Attribute data-name des geklickten edit-Buttons

duplicateFile = true/false; wenn true, wird eine Kopie der App erstellt

Ruft mittels

5.27 newApp(editapp)

das ApplikationTemplate auf, ersetzt die Headline wenn duplikat und bildet mit

[6.26.buildButtonList](#)(appResults.sidebuttons, appBody, 'sidebutton');

[6.26.buildButtonList](#)(appResults.homestation, appBody, 'homestation');

die ggf. vorhandenen Sidebuttons und Homestations ab.

Da evtl. gescrollt werden muss, noch

[6.56.showScroll](#)(\$('.configcenter.full'))

6.36 *editSidebutton(parent, dataFile, dataNum)*

parent = Elternelement des geklickten edit-Buttons

dataFile = Name des Sidebuttons

dataNum = Nummer des Sidebuttons (appResults.sidebuttons[dataNum])

Zuerst wird das SidebuttonTemplate angefordert

[6.28.addSidebutton](#)(false, appResults.sidebuttons[dataNum].iconclass, dataNum)

und dann in die entsprechenden Felder die vorhandenen Werte eingefügt. Zusätzlich werden mit

[6.26.buildButtonList](#)(appResults.sidebuttons[dataNum].tiles, sideButtonBody, 'plants', dataNum);

[6.26.buildButtonList](#)(appResults.sidebuttons[dataNum].alarm_buttons, sideButtonBody, 'alarmbuttons', dataNum);

die Buttons für die Plants und die Alarmbuttons generiert.

Für die Scrollfunktion nochmals

[6.56.showScroll](#)(\$('.configcenter.full'))

6.37 *editPlant(parent, dataSidebuttonNum, dataFile, dataNum)*

parent = Elternelement des geklickten edit-Buttons

dataSidebuttonNum = Nummer des Sidebuttons

dataFile = Name der Plant

dataNum = Nummer der Plant(appResults.sidebuttons[dataNum].tiles[dataNum])

Zuerst wird das PlantTemplate angefordert

5.29 addPlant(false, dataSidebuttonNum,
appResults.sidebuttons[dataSidebuttonNum].tiles[dataNum].iconclass, dataNum)

und mit Daten befüllt. Dann

[6.56.showScroll](#)(\$('.configcenter.full'))

6.38 *editAlarmbutton(parent, dataFile, dataNum, dataSidebuttonNum)*

parent = Elternelement des geklickten edit-Buttons

dataSidebuttonNum = Nummer des Alarmbuttons

dataFile = Name des Alarmbuttons

dataNum = Nummer des Alarmbuttons(appResults.sidebuttons[dataNum].alarm_buttons
[dataNum])

Zuerst wird das AlarmbuttonTemplate angefordert

5.30 addAlarmbutton(false, dataSidebuttonNum,
appResults.sidebuttons[dataSidebuttonNum].alarm_buttons[dataNum].iconclass,
dataNum);

dann mit Daten befüllt und

[6.56.showScroll](#)(\$('.configcenter.full'))

6.39 *editHome(parent, dataFile, dataNum)*

parent = Elternelement des geklickten edit-Buttons

dataFile = Name des Homebuttons

dataNum = Nummer des Homebuttons

Zuerst wird das HomeTemplate angefordert

[6.31.addHome](#)(false, dataNum)

und mit Daten befüllt und

[6.56.showScroll](#)(\$('.configcenter.full'))

6.40 *saveApp(parent)*

parent = die Mutterbox der App

Zuerst wird geprüft, ob alle Pflichtfelder ausgefüllt sind. Ist dem nicht, werde die entsprechenden Felder gekennzeichnet und das erste Fehl-Feld in den Fokus gerückt. Sind alle Felder ausgefüllt. Dann im nächsten Schritt wird geprüft, ob die Applikation evtl. umbenannt wurde. In diesem Fall würde zwar normalerweise die App unter neuem Namen

gespeichert, die „alte“ App wäre aber auch noch vorhanden. Um dies zu verhindern wird also erstmal die alte App gelöscht und bei Erfolg die neue geschrieben. Sollte das Löschen misslingen, wird ein Dialog gezeigt, der den Grund dafür nennt. Wurde der Applikationsname nicht geändert, bekommen die ggf. angelegten Homestations noch eine IconClass verpasst und das ganze wird gespeichert

```
emosWS.writeData(writeJSONAPP + appName + ".json", JSON.stringify(appResults, null, " "));
```

6.41 *activateConfig()*

Die Funktion baut das Config-Overlay, sorgt dafür, dass der Button Config auf active gesetzt wird, baut bei Bedarf den Inhalt des Overlays und öffnet dasselbige mittels Animation. Nach der Öffnung wird dann noch entschieden, welcher Menüpunkt angezeigt werden soll. Ist der Administrator eingeloggt, so wird der Punkt „Server“ angezeigt und die Serverliste mittels

[6.2.GetServerList\(\)](#);

geladen und aufgebaut. Ist ein normaler oder kein User eingeloggt, so wird der Punkt „APP“ gezeigt und die Appliste mittels

[6.24.buildAppList](#)(\$(".configcenter"));

Buttons, die nur für den Administrator bestimmt sind, werden disabled.

6.42 *onOpenDiagnosisWindow(msg)*

Callback für `emosWS.addEventListener("OpenDiagnosisWindow", onOpenDiagnosisWindow)`;

Ruft nur die Funktion

[6.55.onMessage](#)(msg, false);

auf.

6.43 *getDialog(mode, element, num)*

mode = Betriebsart

element = PLCSoftware

num = 1/0 ein- bzw. Ausschalten

Erstellt einen Dialog, um sich einen Betriebsartenwechsel bestätigen zu lassen

6.44 *getDeleteDialog(filename)*

filename = Name der zu löschenden Datei

Erstellt einen Dialog, um sich das Löschen einer Datei bestätigen zu lassen.

6.45 *scrollfunc(e)*

Wird aufgerufen um die Scrollbar zu initiieren, den Scrollbedarf zu errechnen und ggf. bestimmte Bereiche mit einer leichten Transparenz zu versehen:

[6.48.setScrollbar](#)(contentElement);

[6.56.showScroll](#)(\$(contentElement));

[6.47.getTransparenz](#)(\$(contentElement));

6.46 *scrollUpDown(updown, myMainPage, myTarget)*

updown = 'up'/'down' String

myMainPage = Objekt des Mutterelementes oder des Contentelementes

myTarget = Objekt des Elementes(target) beim nutzen des Scrollrades über „unassigned Plants“

Funktion für das Hoch-/Runterscrollen. Es wird zuerst geprüft, welches Element gescrollt werden soll. Dann wird errechnet, wieviel gescrollt und in welche Richtung es gehen soll. Ausgeführt wird das dann von einer Animation, die Schrittweise auch die Position des Scrollbalkens nachstellt.

[6.48.setScrollbar](#)(contentElement);

Nachdem die Animation beendet ist wird errechnet, ob und in welche Richtungen noch gescrollt werden kann und ob einige Elemente mit Transparenz belegt werden sollen.

[6.56.showScroll](#)(\$(contentElement));

[6.47.getTransparenz](#)(contentElement);

6.47 *getTransparenz(contentElement)*

contentElement = das Mutterelement in dem sich z.B. die Kacheln befinden

Für die Kacheln der Anlagen und den Boxen der Anlagen im Alarmoverlay gilt, wenn sie außerhalb eines bestimmten Bereiches liegen, sollen sie zu 50% transparent sein.

6.48 *setScrollbar(contentElement)*

contentElement = Objekt des Mutterelementes welches den Scrollbalken erhält

Setzt den Scrollbalken in Relation zum gescrollten Element.

6.49 *scrollLeftRight(direction, nonDirection, nextVisibleElement, nextElement, newPosition, self)*

direction = 'left'/'right' String für die Scrollrichtung

nonDirection = 'left'/'right' String für die Scrollgegenrichtung

nextVisibleElement = Objekt des Elements, welches gezeigt werden soll

nextElement = Objekt des Elementes das vor bzw nach nextVisibleElement kommt

newPosition = 500/-500; für die rein-raus-Animation

self = der Button, auf den geklickt wurde

Steuert das Blättern durch die SubTabs in den Statusfenstern und sorgt auch für die richtige Anzeige der „Seitenzahl“ im Footer.

6.50 *getId()*

Erzeugt eine ID

6.51 *clearMyInterval(self)*

Beendet einen bestimmten oder eine Liste von laufenden Intervallen.

6.52 *getAnimationData(name, tile)*

name = Sidebuttonname, String

tile = Array von Objekten der Tiles

Erzeugt ein Objekt mit Daten, die für das gleichzeitige Starten der Alarmanimationen benötigt werden.

6.53 *ulMaker(myClass, myHtml, told, myID)*

myClass = Klasse und ID(wenn kein myId) des UL-Elements

myHtml = Array mit den -Elementen als String

told = ID des Mutterelements, welches die UL aufnimmt

myId = ID falls anders als Klassenname

Erzeugt aus den empfangenen Daten eine unsortierte Liste, die für Menüs, DropDown etc genutzt werden.

6.54 *scrolltabs(self)*

self = der geklickte Button

Wenn in den Statusfenstern mehr als 3 Tabs vorhanden sind, wird ein Scrollbutton für die Tabs eingeblendet. Diese Funktion steuert das Verhalten dieses Buttons, verschiebt die Tabs, ändert die Scrollrichtung etc.

6.55 *onMessage(msg, anotherWin)*

msg = die weitergeleitete Serverantwort des

emosWS.addEventListener("OpenDiagnosisWindow", onOpenDiagnosisWindow);

anotherWin = true/false; wenn true wird ein weiteres Statusfenster geöffnet

6.56 *showScroll(pageElement)*

6.57 *animateOff(layer, toHide, activeParent, notclose)*

Layer = ID oder Klasse des zu schließenden Elementes

toHide = ID oder Klasse des zu schließenden Elementes

activeParent = ID des Mutterelements

notclose =

Schließt ein geöffnetes Overlay und bei Bedarf werden auch noch Intervall und aktive AlarmClients geschlossen

[6.51.clearMyInterval](#);

[6.88.terminateActiveAlarmClient](#)();

6.58 *getHelpObject(cCode)*

cCode = CountryCode

Mithilfe des countryCode wird per

emosWS.rest.framework.getFileInfo

der dem countryCode entsprechende Ordner unter dem Ordner „help“ ausgelesen.

Existieren in dieser Sprache keine Hilfe-Dateien, so wird ein Hinweis eingeblendet der einen Link enthält, um die englischen Hilfe-Files zu laden. Sind Daten vorhanden(ggf. Auch nach der Link-Betätigung um die englischen zu laden), werde diese an die Funktional

[6.59.buildHelpLinks](#)(helpResults);

gesendet.

6.59 *buildHelpLinks(helpResults)*

helpResults = Objekt mit dem Inhalt des entsprechenden Ordners

Für jeden Eintrag in helpResults wird ein Span-Element erzeugt, welches dann über eine click-Funktion verfügt, die per ajax das entsprechende PDF-Dokument lädt und anzeigt.

6.60 *closeOther(me)*

me = ID des geklickten Elements

Da von den Overlays, die hinter den Icons im Header liegen, immer nur eines geöffnet sein darf, muss beim Öffnen eines anderen Overlays ein ggf. offenes geschlossen werden. Im Falle des AlarmOverlays müssen noch zusätzliche Funktionen ausgeführt werden.

[6.64.alarmheaderReset](#)();

[6.51.clearMyInterval](#)();

[6.88.terminateActiveAlarmClient](#)();

Die anderen Overlays werden dann mit

[6.57.animateOff](#)(layer, toHide, activeParent, notclose)

geschlossen.

6.61 *buildHeader(pageld)*

pageld = Objekt der aktuell sichtbaren Tab-Page

Diese Funktion bildet zum Einen die Seitenanzeige und dessen Dropdown-Menü im Footer(anfänglich war dies noch Teil des Headers) und zum Anderen wird die Überschrift gesetzt. Die Überschrift wird in ein input-Feld gesetzt, welches aber nur bei den private-Tab im eingeloggten Zustand bearbeitbar ist.

6.62 scrollUpDownButton(where, parent, line, updown)

where = String ID/Klasse; hinter welchem Element sollen die Buttons gesetzt werden

parent = String ID/Klasse in welches Element sollen die Buttons gesetzt werden

line = obsolet?

updown = String scrollDown/scrollUp – Css-Klasse

Erstellt scrollUp/scrollDown Buttons

6.63 buildTiles(appNum, appName)

appNum = Nummer des Sidebuttons zu dem die Tiles gehören

appName = Name des Sidebuttons zu dem die Tiles gehören

Mit Hilfe der appNum wird der entsprechende Sidebutton als aktiv gekennzeichnet und aus dem „results“-Objekt die zugehörigen Tiles gelesen. Dann werden in eine Schleife die Kacheln des Contentbereichs und des Alarmheader aufgebaut. Bei geöffnetem Alarmheader-Overlay sieht am bekanntlich ein Liste sämtlicher Anlagen mit Buttons für einen Betriebsartenwechsel. In dieser Schleife wird nun die Anlagenkachel mit der Checkbox gebaut. Erster Durchlauf von

[6.65.buildAlarmHeader](#)(appNum, appName, addAnimationsJobs, alarmTiles, true, val, key)

Die Kacheln werden mit einem Alarmflag versehen und advised. Es werden zusätzlich noch Scrollbuttons eingefügt

[6.62.scrollUpDownButton](#), einen für up, einen für down.

Dann wird überprüft, ob eine oder mehrere Kachen ausserhalb eines bestimmten Bereiches liegen. Wenn ja, werden die Scrollbuttons angezeigt und diese Kachen mit einer 50%igen Transparenz versehen.

Dann wird erneut die

[6.65.buildAlarmHeader](#)(appNum, appName, addAnimationsJobs)

aufgerufen, um die restlichen Buttons und den Mega-Button des Alarmheader zu setzen. Bei Bedarf werden noch

scrollUp/-DownButtons [6.62.scrollUpDownButton](#),

DetailsButton und Details-Reset-Buttons gesetzt. Der DetailsButton führt auf Klick die

[6.66.getDetailsOverlay\(\)](#) aus, vorausgesetzt, es wurde min. eine Anlage vorher ausgewählt. Ist keine Anlage ausgewählt, ist dieser Button inaktiv.

Der Details-Reset-Button wird nur auf dem DetailOverlay sichtbar und resettet auf Klick die Anlagenauswahl durch

[6.64.alarmheaderReset\(\)](#)

6.64 alarmheaderReset()

Diese Funktion sorgt für die Deaktivierung des DetailReset-Button, die Wiederanzeige sämtlicher Anlagen-Kacheln, entfernt die Reports aus dem DOM, hebt die Anlagenauswahl auf und entfernt nicht mehr benötigtes. Es wird überprüft, ob einige Kachel eine Transparenz benötigen

[6.47.getTransparenz](#)(\$('#alarmpage_content'))

und ob der Scrollbalken benötigt wird

[6.56.showScroll](#)(\$('#alarmpage_content'))

6.65 *buildAlarmHeader(appNum, appName, addAnimationsJobs, alarmTiles, neededforAlarmPage, tile, num)*

appNum = Nummer des Sidebuttons zu dem die Tiles gehören

appName = Name des Sidebuttons zu dem die Tiles gehören

addAnimationsJobs = Array

alarmTiles = Array mit den Buttons für die Anlagen

neededforAlarmPage = true/false; wenn true wird eine andere ID für die Alarmanimation übergeben.

tile = Datenobjekt für die Kachel

num = Tilenummer

In einer Schleife werden mit den Alarmbuttons, die den Sidebuttons in der app.json zugeordnet sind, Listenelement erstellt, die dann aufgrund ihres Buttontypes einen Singlebutton oder Blockbutton bilden und dann im Header-Bereich sichtbar sind. Wenn das Array alarmTiles gefüllt ist, werden die Buttons für die Anlagen, die bei geöffnetem AlarmOverlay angezeigt werden, gebildet. Anschließend wird noch die PLC für die Alarmanimation angelegt.

6.66 *getDetailsOverlay()*

Hier werden die Blöcke gebildet, die nach der Anlagenauswahl im Alarmheader, rechts angezeigt werden, also die Reports, Trending und Private. Dafür werden zuerst die PLC's der Anlagen gesammelt, gefolgt von eig. Aufbau der einzelnen Blöcke, die dann die Report-Tabellen aufnehmen und anzeigen. Es wird auch ein zusätzlicher Table-Header gebaut. Der von der emos gelieferte Table-Header wird ausgeblendet. Dadurch bleibt dann der Header fix beim scrollen.

Jeder Block reagiert auf Klick, um dann Full-Size angezeigt zu werden.

[6.69.resizeDetail](#)(\$(this))

Damit die Report-Tabellen ausgeliefert werden, wird über die emos eine entsprechende Anfrage gesendet: (Beispiel für einen Block)

```
emosWS.rest.alarm.topOfType({
  type: "alarming",
  alarmGroup: [myPLCs], //myPLCs '$System'
  success: function(alarmArray){
    new emosWS.RestAlarmClient($('#.detailAlarmingTable'), alarmArray,
onRestAlarmReady, 'Alarming');
  },
  error: function(){
    console.log("request failed");
  }
}
```

```
});
```

Als Antwort erhalten wir dann die gewünschte Tabelle.

Die Blöcke Trending und Private werden erst nach Klick mit Daten befüllt.

6.67 *onRestAlarmReady(motherTable)*

motherTable = string; Headline der Report-Box

Das hier ist das Callback für die emos-Anfrage der Report-Tabellen. Durch motherTable können wir die benötigten Daten aus dem „filters“-Array, das headline-Element und die darauf foldende Tabelle identifizieren und senden diese Daten an

[6.68.doFilter](#)(filters[motherTable], elem.nextAll('.DiagnosisRestAlarmTableHead'), motherTable);

Aufgrund der Filterung(oder nicht-Filterung) wird dann noch angegeben, wie viele Datenzeilen von der Gesamtzahl der geladenen Zeilen sichtbar sind und die Scrollbuttons gesetzt

[6.56.showScroll](#)(elem.nextAll('.detailTable'));

6.68 *doFilter(filters, listelement, motherTable)*

filters = Array mit den Button-Objekten der gewählten Filter

listelement = Objekt des jeweiligen Tabellenheaders

motherTable = string; Headline der jeweiligen Report-Box

Hier wird im Grunde nur überprüft, ob die Liste gefiltert werden muss, also wenn in „filters“ Daten vorhanden sind, wird in einer Schleife pro Filterelement

[6.75.filterMyList](#)(listelement, \$(val).attr('data-name'), \$(val).attr('data-value'), motherTable);
aufgerufen.

6.69 *resizeDetail(self)*

self = die geklickte Box

Sorgt dafür, dass die Report-Boxen die volle Größe annehmen. Wurde die Trending-Box geklickt, wird der benötigte Container erschaffen. Ist der User eingeloggt und hat in seinen Einstellungen ein Trending-Favoriten gespeichert, wird dieser in einem neuen Trending geladen. Ansonsten wird ein leeres Trending geladen, aber mit der Einladung, ein Trendingset zu öffnen.

Wurde die Private-Box geklickt, werden mittels

[6.80.togglePrivPub](#)('public'); oder [6.80.togglePrivPub](#)('private'); je nachdem ob der User eingeloggt ist, die private- oder publicTabs geladen.

Wurde eine der 3 Report-Boxen geklickt, wird ein Dropdown für die Anzahl der zu ladenden Einträge und ein Reload-Button angelegt. Ist die Tabelle dann geladen, werden die Scrollelemente gezeigt

[6.56.showScroll](#)(self.find('.detailTable'));

und die Filteroverlays in den entsprechenden Tabellenspalten gebildet. z.B.

```
// context for filter priority
```

```

$('<ul/>', {
    "class": "overlay sortPriority"
}).appendTo('.detail'+ actHead + ' .DiagnosisRestAlarmTableHead .tdPriority');
var filterElements = unique( self.find('.detailTable')[0][actHead +
'tdPriority']).sort();
6.73.buildFilterOverlay(filterElements, 'tdPriority', '.sortPriority', self);

```

6.70 *reloadTables()*

Diese Funktion lädt alle 3 Repot-Tabellen neu, damit die neuen Daten vorliegen. Nach dem Laden werden diese neuen Daten auch nach den eingestellten Filtern gefiltert.

6.71 *preparePrivateTabs(myval, mykey, toggleme, myText)*

myval = string; Name des Tabs

mykey = index des Tabs in privateResults

toggleme = obsolete?

myText = string; Private oder Public

Da die PrivateTabs im AlarmOverlay etwas anders dargestellt werden, wird hier das benötigte Datenobjekt präpariert und über die emos aufgerufen

```

new emosWS.HTMLDiagnosisWindow(myMsg, onPrivateReady, null, '.windowHolder',
switcher);

```

6.72 *onPrivateReady(switcher)*

switcher = obsolete?

Callback des emos-Aufrufes für die private-/publicTabs im Overlay. Sorgt für die korrekte Positionierung der Tabs, versteckt nicht benötigte Elemente, zeigt Scrollelemente

[6.56.showScroll](#)(pageld);

und sorgt für Headline und Footer des Tabs

[6.61.buildHeader](#)(pageld);

6.73 *buildFilterOverlay(filterElements, tdName, parent, self)*

filterElements = Array mit den Filterelementen

tdName = Ziel, Tabellenspalte die das Overlay trägt

parent = die Klasse der benötigten UL

self = Mutterelement welches die Tabelle trägt

Das FilterOverlay besteht aus Listenelementen mit click-Funktion, die in einer Schleife aufgebaut werden. Trägt das Li die Klasse „active“ so sorgt der Klick dafür, dass der entsprechende Filter entfernt wird

[6.76.removeFilter](#)(\$

```
(this).closest('.DiagnosisRestAlarmTableHead').prev().find('.filtertext[data-value='+ val +']');
```

ansonsten wird der Filter hinzugefügt

```
6.75.filterMyList($(this).closest('.DiagnosisRestAlarmTableHead'), tdName, val, $(this).closest('.DiagnosisRestAlarmTableHead').prevAll('.detailHead').text());
```

Nach dem die Schleife durchlaufen ist, wird am Ende noch jeweils ein Resetter angefügt.

```
6.74.resetter(tdName.replace('td', '').toLowerCase() + 'All', '.' + tdName, '' + parent, self);
```

6.74 *resetter(addClass, tab, parent, self)*

addClass = zusätzliche Klasse für das Reset-Listenelement

tab = Tabellenspalte

parent = Klasse der UL, dort wird der Resetter dann eingefügt

self = Mutterelement welche die Tabelle hält

Auf click werden alle gesetzten Filter zurückgesetzt.

6.75 *filterMyList(self, tabClass, filterval, motherTable)*

self = der aktuelle TabellenHeader

tabClass = Tabellenspalte deren Filter genutzt wurde

filterval = der Filter

motherTable = die entsprechende Tabelle

Zuerst wird überprüft, ob aus der Gesamtliste oder von einer bereits gefilterten Liste ausgegangen wird. Dementsprechend wird eine Liste angelegt, die dann zu filtern ist. In einer Schleife wird dann jede Tabellenzeile, die sichtbar sein soll, in ein Objekt-Array geschrieben. Dann wird ein Filterbutton aufgebaut, der anzeigt, wonach gefiltert wurde. Das Objekt-Array wird mit der Original-List abgeglichen und nur die Zeilen angezeigt, die benötigt werden. Danach werden noch in einer Schleife die Filteroverlays aktualisiert

```
6.73.buildFilterOverlay(filterElements, key, tmpTableElements[key], self.parent());
```

danach die Anzahl der gefilterten Elemente und die Scrollelement aktualisiert.

6.76 *removeFilter(self)*

Entfernt den geklickten Filter und baut die Liste entsprechend neu auf.

6.77 *unique(array)*

Entfernt doppelte Einträge in einem Array

6.78 *closeDetail(self)*

Zum schließen eines geöffneten Report-Overlays. Dazu werden benötigte Elemente wieder eingeblendet, bzw. ausgeblendet oder aus dem DOM entfernt.

6.79 *saveHeadline(self)*

Die Überschriften der Private-Tabs können geändert werden. Wenn also ein User in die Headline geklickt und diese geändert hat, wird die neue Headline im Datenobjekt der user.json geändert und auf dem Server gespeichert.

6.80 *togglePrivPub(self, myid)*

Funktion um zwischen Private-Tabs und Public-Tabs zu toggeln. Je nach Auswahl wird dann per

[6.81.getPrivateTabs\(preparent, forOverlay, toggleme, private\)](#)

entweder die Liste der privates aus der user.json oder der publics aus der public.json geladen. Wenn die Umschaltung innerhalb des Private-Overlay geschieht, wird dann noch der Footer für die Privates/Publics benötigt

[6.82.getPrivateTabFooter\(\)](#)

6.81 *getPrivateTabs(preparent, forOverlay, toggleme, private)*

preparent = Muttercontainer, die die Inhalte hält

forOverlay = true/false;

toggleme = true/false; dient zur umschaltung zw. Privat und public

private = string;

Zuerst wird unterschieden ob der User eingeloggt ist und ob er bereits Private-Tabs hat. Entsprechend wird ein Array vorbereitet, welches zum Aufbau der Tabs benötigt wird. Auch wird der Toggle-Text festgelegt(show private/show public). Dann wird in einer Schleife für jeden zu ladenden Tab benötigte Einstellungen vorgenommen. Soll der Tab z.B. im Overlay platziert werden, muss das Datenobjekt, welches für den eigentlichen Aufbau benötigt wird, speziell etwas angepasst werden durch

[6.71.preparePrivateTabs\(val, key, toggleme, tempText\);](#)

Dann

6.82 *getPrivateTabFooter()*

6.83 *actionForSave(e)*

Werden im Statusfenster Änderungen vorgenommen, sei es mit der Maus, der Tastatur oder der virtuellen Tastatur werden diese Änderungen ausgelesen und per emos gespeichert. Dann wird die

[6.84.actionForCancel\(e\)](#)

ausgeführt, um ggf. Elemente zu destroyen und/oder zu verstecken

6.84 *actionForCancel(e)*

Versteckt Elemente und zerstört die von PlugIns gelieferten Elemente (Datepicker etc)

6.85 *loadIframe(e, alarmheader)*

6.86 *onSiteProperties(msg)*

6.87 *activateButton()*

Die beiden Quickcontrolbuttons oberhalb des Statusfensters dürfen nur mit entsprechenden Rechten betätigt werden. Sind diese Rechte nicht vorhanden, werden die Buttons deaktiviert.

6.88 *terminateActiveAlarmClient()*

Ein aktiver AlarmClient wird terminiert, um Ressourcen wieder freizugeben

6.89 *addTrendPage(parent, myPlc, showControl)*

6.90 *buildAlarmList()*

6.91 *changeActive(self, showme)*

6.92 *listen_again()*