to:Huus

Quellcode

Browser: Chrome, OS: Windows, DB: config/dbConf.json, Startprozedur: Main.go

Dokumentation zur Hausarbeit für das
Modul Webprogrammierung an der Hochschule Flensburg

von **Sven Kuhlmann**

MatrikelNr: 610292
Student: Sven Kuhlmann
Geboren am: 09.01.1994 in Leer
Durchführungszeitraum: 15.11.2017 – 20.12.2017

WebProg

Project:	to:Huus	Datum:	20.12.2017
Dok.-Typ:	Quellcode	Name:	Sven Kuhlmann
Version:	1.03	Matr.-Nr:	610292

MAIN.GO

```
// Main for toHuus
// Call this to start the application
// Just this need to be called, the simulation will start by it self
// Package db, controllers, simulator is needed
package main

import (
	"fmt"
	"net/http"
	"toHuus/controllers"
	"toHuus/db"
	"toHuus/simulator"
)

// Main function for this application
// This need to be run
func main() {
	// Initialise DB from config
	db.Database(controllers.DbConfig())
	// This delete the db to start clean
	//db.OpenConnection().DropDatabase()
	// Preset types
	controllers.SetDefaultTypes()
	// Preset simulator states
	controllers.SetSimStates()
	// Start simulator as thread
	go simulator.Start() // Interoperability
	// Handler
	http.Handle("/images/", http.FileServer(http.Dir("./toHuus/views/")))
	http.Handle("/assets/", http.FileServer(http.Dir("./toHuus/views/")))
	http.Handle("/avatar/", http.FileServer(http.Dir("./toHuus/conf/")))
	http.HandleFunc("/", controllers.CheckLogin)
	http.HandleFunc("/ui", controllers.InterfaceHandler)
	http.HandleFunc("/ui/user", controllers.UserHandler)
	http.HandleFunc("/ui/add", controllers.AddHandler)
	http.HandleFunc("/ui/get", controllers.GetHandler)
	http.HandleFunc("/ui/del", controllers.DelHandler)
	http.HandleFunc("/ui/set", controllers.StateHandler)
	http.HandleFunc("/sim", controllers.SimulatorHandler)
	http.HandleFunc("/sim/data", controllers.DataHandler)
	http.HandleFunc("/sim/get", controllers.SimGetHandler)
	http.HandleFunc("/sim/set", controllers.SimSetHandler)
	err := http.ListenAndServe(":4242", nil)
	if err != nil {
		fmt.Println(err)
	}
}
```

SIMULATOR.HTML

```
{{ define "content" }}
<!-- Article -->
  <!-- Sim -->
  <section id="simUi" class="one dark cover">
    <div class="container">
      <article>
        <h3>Current</h3>
        <header>
        </header>
        <h3>Actions</h3>
        <form class="smaller">
          Set state: <input id="simStateOn" type="button" value="On">
          <input id="simStateOff" type="button" value="Off"><br>
          Set current time: <input id="simTime" type="time"><br>
          Set zoom factor (Multiplier): <input id="simMultiplier" min="1" max="100" value="1" type="range"><br>
        </form>
      </article>
    </div>
  </section>
  <!-- Data -->
  <section id="data" class="four">
    <div class="container">
```

```
        <header>
          <h2>Data</h2>
        </header>
        <form method="post" action="/sim/data?type=export">
          <fieldset>
            <span><b>Export data by XML: </b></span><br>
            <input id="exportBtn" type="submit" name="submit" value="Export">
          </fieldset>
        </form>
        <br><br>
        <form method="post" action="/sim/data?type=import" enctype="multipart/form-data">
          <fieldset>
            <span><b>Import data by XML: </b></span><br>
            <input type="hidden" name="test" value="test">
            <input type="file" name="dataFile" id="dataFile" multiple="multiple">
            <input type="submit" name="submit" id="dataFileSub" value="Import" disabled>
          </fieldset>
        </form>
      </div>
    </section>
    <!-- About -->
    <section id="about" class="three">
      <div class="container">
        <header>
          <h2>About</h2>
        </header>
        <p>Bei dieser Webseite handelt es sich um eine Implementierung eines SmartHomes.
          Da kein echtes System angebunden ist, wurde ein Simulator integriert.</p>
        <p>Diese Webseite wurde im Rahmen einer Hausarbeit der Hochschule Flensburg erschaffen.
          Sie ist Inhalt der Prüfungsleistung für das Modul Webprogrammierung und wurde in alleiniger Arbeit von
          von <b>Sven Kuhlmann</b> erstellt.<br>
          Sämtliches wissen zum erstellen des Inhaltest stammt aus den Vorlesungen/Laboren und dem eigenen
          Wissenstand.<br>
          Da für einige Medien eventuell keine Rechte bestehen, ist der öffentliche Gebrauch untersagt.</p>
        <p>Für weitere Informationen oder Fragen kontaktieren Sie mich hier:<br>
          <a href="mailto:Sven.Kuhlmann@stud.hs-flensburg.de">EMail senden</a></p>
      </div>
    </section>
<!-- Helper -->
    <!-- Logout
    <form method="post" action="../">
      <input title="Logout" type="image" src="images/logout.png" alt="Logout" id="logout" name="authBtn" value="Logout">
    </form>
    <!-- Interface
    <form method="post" action="/ui">
      <input title="Interface" type="image" src="images/interface.png" alt="Interface" id="sim" name="sim" value="Interface">
    </form> -->
{{ end }}
```

LOGIN.HTML

```
{{ define "content" }}
<!-- Article -->
  <!-- Login -->
  <section id="login" class="one dark cover">
    <div class="container">
      <header>
        <h3 class="alt">Hallo! Ich bin <strong>to:Huus</strong>, ein Smart-Home Steuersystem</h3>
        <h2>Logge dich erst ein, um mich zu nutzen!</h2><br>
        <form method="post" action="">
          <div class="row centered">
            <input type="text" name="uname" placeholder="Username" />
            <input type="password" name="passwd" placeholder="Password" />
          </div>
          <div class="row centered">
            <input type="submit" name="authBtn" value="Login" />
            <input type="submit" name="authBtn" value="Registration" />
          </div>
        </form>
      </header>
      <footer></footer>
    </div>
  </section>
  <!-- Data -->
  <section id="data" class="four">
    <div class="container">
      <header>
        <h2>Data</h2>
      </header>
      <form method="post" action="/sim/data?type=export">
        <fieldset>
          <span><b>Export data by XML: </b></span><br>
          <input id="exportBtn" type="submit" name="submit" value="Export" disabled>
        </fieldset>
      </form>
      <br><br>
      <form method="post" action="/sim/data?type=import" enctype="multipart/form-data">
        <fieldset>
          <span><b>Import data by XML: </b></span><br>
          <input type="hidden" name="test" value="test">
          <input type="file" name="dataFile" id="dataFile" multiple="multiple">
          <input type="submit" name="submit" value="Import">
        </fieldset>
      </form>
    </div>
  </section>
  <!-- About -->
  <section id="about" class="three">
    <div class="container">
      <header>
        <h2>About</h2>
      </header>
      <p>Bei dieser Webseite handelt es sich um eine Implementierung eines SmartHomes.
        Da kein echtes System angebunden ist, wurde ein Simulator integriert.</p>
      <p>Diese Webseite wurde im Rahmen einer Hausarbeit der Hochschule Flensburg erschaffen.
        Sie ist Inhalt der Prüfungsleistung für das Modul Webprogrammierung und wurde in alleiniger Arbeit von
        von <b>Sven Kuhlmann</b> erstellt.<br>
        Sämtliches wissen zum erstellen des Inhaltest stammt aus den Vorlesungen/Laboren und dem eigenen
        Wissenstand.<br>
        Da für einige Medien eventuell keine Rechte bestehen, ist der öffentliche Gebrauch untersagt.</p>
      <p>Für weitere Informationen oder Fragen kontaktieren Sie mich hier:<br>
        <a href="mailto:Sven.Kuhlmann@stud.hs-flensburg.de">EMail senden</a></p>
    </div>
  </section>
{{ end }}
```

INTERFACE.HTML

```
{{ define "content" }}
<!-- Article -->
    <!-- Overview -->
    <section id="home" class="one dark cover">
```

```
        <div class="container">
            <article>
            </article>
        </div>
    </section>
    <!-- Devices -->
    <section id="devices" class="three">
        <div class="container">
            <header>
                <h2>Devices</h2>
            </header>
            <article>
                <table>
                    <tr>
                        <th>Name</th>
                        <th>Room</th>
                        <th>Type</th>
                        <th>Action</th>
                    </tr>
                {{ range .Devices }}
                    <tr id="D_{{ .Name }}">
                        <td>{{ .Name }}</td>
                        <td>{{ .Room }}</td>
                        <td>{{ .Type }}</td>
                        <td>
                            <img src="images/edit.png" class="edit-btn" name="DeviceItemEdit" id="D_{{ .Name }}" />
                            <img src="images/delete.png" class="delete-btn" name="DeviceItemDel" id="D_{{ .Name }}" />
                        </td>
                    </tr>
                {{ end }}
                </table>
            </article>
        </div>
    </section>
    <!-- Events -->
    <section id="events" class="three">
        <div class="container">
            <header>
                <h2>Events</h2>
            </header>
            <article>
                <table>
                    <tr>
                        <th>Name</th>
                        <th>Time</th>
                        <th>Offset</th>
                        <th>Devices(State)</th>
                        <th>Action</th>
                    </tr>
                {{ $events := .Events }} {{ $rels := .Rel }}
                {{ range $event := $events }}
                    <tr id="E_{{ $event.Name }}">
                        <td>{{ $event.Name }}</td>
                        <td>{{ $event.Time }}</td>
                        <td>{{ $event.Offset }}</td>
                        <td>{{ range $rel := $rels }}{{ if eq $event.Id $rel.Id }}<span class="hide">{{ $rel.Room }}|</span>{{ $rel.Name }}({{ $rel.Value }}), {{ end }}{{ end
}}</td>
                        <td>
                            <img src="images/edit.png" class="edit-btn" name="EventItemEdit" id="E_{{ $event.Name }}" />
                            <img src="images/delete.png" class="delete-btn" name="EventItemDel" id="E_{{ $event.Name }}" />
                        </td>
                    </tr>
                {{ end }}
                </table>
            </article>
        </div>
    </section>
    <!-- Types -->
    <section id="types" class="two">
        <div class="container">
            <header>
                <h2>Types</h2>
            </header>
            <article>
                <table>
                    <tr>
                <th>Name</th>
                <th>Kind</th>
                <th>Min</th>
                <th>Max</th>
                <th>Action</th>
              </tr>
            {{ range .Types }}
              <tr id="T_{{ .Name }}">
                <td>{{ .Name }}</td>
                <td>{{ .Kind }}</td>
                <td>{{ .Min }}</td>
                <td>{{ .Max }}</td>
                <td>
                  <img src="images/edit.png" class="edit-btn" name="TypeItemEdit" id="T_{{ .Name }}" />
                  <img src="images/delete.png" class="delete-btn" name="TypeItemDel" id="T_{{ .Name }}" />
                </td>
              </tr>
            {{ end }}
            </table>
          </article>
        </div>
      </section>
      <!-- User -->
      <section id="user" class="four">
        <div class="container">
          <header>
            <h2>User</h2>
          </header>
          <form method="post" action="/ui/user?set=title">
            <fieldset>
              <span><b>Set a user title: </b></span><br>
              <input type="text" name="title" id="titleIpt">
              <input type="submit" name="submit" value="Set">
            </fieldset>
          </form>
          <form method="post" action="/ui/user?set=avatar" enctype="multipart/form-data">
            <fieldset>
              <span><b>Set an avatar image: </b></span><br>
              <input type="hidden" name="test" value="test">
              <input type="file" name="avatarFile" id="avatarFile" multiple="multiple">
              <input type="submit" name="submit" value="Upload">
            </fieldset>
          </form>
          <form method="post" action="/ui/user?del=user">
            <fieldset>
              <span><b>Delete this user: </b></span><br>
              <input type="submit" name="submit" value="Delete">
            </fieldset>
          </form>
        </div>
      </section>
      <!-- About -->
      <section id="about" class="three">
        <div class="container">
          <header>
            <h2>About</h2>
          </header>
          <p>Bei dieser Webseite handelt es sich um eine Implementierung eines SmartHomes.
            Da kein echtes System angebunden ist, wurde ein Simulator integriert.</p>
          <p>Diese Webseite wurde im Rahmen einer Hausarbeit der Hochschule Flensburg erschaffen.
            Sie ist Inhalt der Prüfungsleistung für das Modul Webprogrammierung und wurde in alleiniger Arbeit von
            von <b>Sven Kuhlmann</b> erstellt.<br>
            Sämtliches wissen zum erstellen des Inhaltest stammt aus den Vorlesungen/Laboren und dem eigenen
            Wissenstand.<br>
            Da für einige Medien eventuell keine Rechte bestehen, ist der öffentliche Gebrauch untersagt.</p>
          <p>Für weitere Informationen oder Fragen kontaktieren Sie mich hier:<br>
            <a href="mailto:Sven.Kuhlmann@stud.hs-flensburg.de">EMail senden</a></p>
        </div>
      </section>
<!-- Helper -->
      <!-- Device Dialog -->
      <dialog id="addDevice" class="dialog">
        <form method="post" action="/ui/add">
        <header class="dialog-header">
          <h1>Add a device</h1>
        </header>
```

```
  <div class="dialog-content">
    <table>
      <tbody>
      <tr>Name: <input type="text" name="dName" placeholder="Name"></tr>
      <tr>Room: <input type="text" name="dRoom" placeholder="Room"></tr>
      <tr>Type:
        <select title="Type" name="dType">
        {{ range .Types }}
          <option name="{{ .Name }}">{{ .Name }}</option>
        {{ end }}
        </select>
      </tr>
      </tbody>
    </table>
  </div>
  <div class="btn-group">
    <input type="submit" class="btn btn-add" id="addD" name="addDevice" value="Add" disabled>
    <input type="reset" class="btn btn-cancel" id="cancelD" value="Cancel">
  </div>
  </form>
</dialog>
<!-- Event Dialog -->
<dialog id="addEvent" class="dialog">
  <form method="post" action="/ui/add">
  <header class="dialog-header">
    <h1>Add an event</h1>
  </header>
  <div class="dialog-content">
    <table id="eDevices-table">
      <tbody>
      <tr><td colspan="2">Name: <input type="text" name="eName" placeholder="Name"></td></tr>
      <tr>
        <td>Time: <input type="time" name="eTime" placeholder="Time"></td>
        <td>Offset(min): <input type="number" id="eOffset" name="eOffset" min="-10" max="10" placeholder="0"></td>
      </tr>
      <tr><td colspan="2"><hr>Item | Set to<button id="eDevices-btn" type="button" class="addItem">+</button></td></tr>
      <tr class="item" name="eDevices-tr" id="eDevices1"><td>
        <select title="Devices" name="eDevice">
          <optgroup label="Devices">
          {{ range .Devices }}
            <option name="{{ .Name }}">{{ .Name }} | {{ .Room }}</option>
          {{ end }}
          </optgroup>
        </select>
        <input type="number" class="fromTo" name="to" min="" max="" placeholder="To">
      </td></tr>
      </tbody>
    </table>
  </div>
  <div class="btn-group">
    <input type="submit" class="btn btn-add" id="addE" name="addEvent" value="Add" disabled>
    <input type="reset" class="btn btn-cancel" id="cancelE" value="Cancel">
  </div>
  </form>
</dialog>
<!-- Type Dialog -->
<dialog id="addType" class="dialog">
  <form method="post" action="/ui/add">
    <header class="dialog-header">
      <h1>Add a type</h1>
    </header>
    <div class="dialog-content">
      <table id="tDevices-table">
        <tbody>
        <tr>Name: <input type="text" name="tName" placeholder="Name"></tr>
        <tr>Kind:
          <select title="Kind" name="tKind" id="tKind">
            <option name="switch">Switch</option>
            <option name="range">Range</option>
            <option name="number">Number</option>
          </select>
        </tr>
        </tbody>
      </table>
    </div>
    <div class="btn-group">
```

```
          <input type="submit" class="btn btn-add" id="addT" name="addType" value="Add" disabled>
          <input type="reset" class="btn btn-cancel" id="cancelT" value="Cancel">
        </div>
      </form>
    </dialog>
    <!-- Add -->
    <input type="checkbox" id="add" class="add"/>
    <label title="Add" class="addLabel" for="add"></label>
    <div id="addMenu">
      <ul id="addItems">
        <li id="addDeviceBtn"><a href="#devices">Device</a></li>
        <li id="addEventBtn"><a href="#events">Event</a></li>
        <li id="addTypeBtn"><a href="#types">Type</a></li>
      </ul>
    </div>
    <!-- Logout -->
    <form method="post" action="../">
      <input title="Logout" type="image" src="images/logout.png" alt="Logout" id="logout" name="authBtn" value="Logout">
    </form>
    <!-- Simulator -->
    <form method="post" target="_blank" action="/sim">
      <input title="Simulator" type="image" src="images/sim.png" alt="Simulator" id="sim" name="sim" value="Simulator">
    </form>
{{ end }}
```

HEADER.HTML

```
{{ define "header" }}
<!DOCTYPE html>
<html lang="en">
<head>
  <title>to:Huus</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, height=device-height, initial-scale=1" />
  <link rel="shortcut icon" type="image/x-icon" href="images/favicon.ico">
  <link rel="stylesheet" type="text/css" href="assets/css/main.css" />
  <script language="javascript" type="text/javascript" src="assets/js/main.js"></script>
</head>
<body>
<!-- Header -->
  <div id="headerToggle">
    <a href="#header" class="toggle"></a>
  </div>
  <div id="header">
    <div class="top">
      <!-- Logo -->
      <div id="logo" title="{{ .User.Username }}">
        <span class="image avatar48"><img id="avatar" src={{ if .User.Avatar }}"{{ .User.Avatar }}?s={{ .User.SessionId }}"{{ else }}"images/avatar.jpg"{{ end }}
name="{{ .User.Username }}" alt=""/></span>
        <h1 id="title" >{{ .User.Username }}</h1>
        <p>{{ .User.Title }}</p>
      </div>
      <!-- Nav -->
      <nav id="nav">
        <ul>
          {{ range .Nav.Elements }}
            <li><a href="#{{ .Ref }}" id="{{ .Ref }}-link"><span class="icon fa-{{ .Icon }}">{{ .Name }}</span></a></li>
          {{ end }}
        </ul>
      </nav>
    </div>
  </div>
<!-- Main -->
  <div id="main">
    <!-- Message -->
    <div id="message">
      <span>{{ .Message }}</span>
    </div>
    <!-- Content -->
{{ end }}
```

FOOTER.HTML

```
{{ define "footer" }}
```

```
    </div>
    </body>
</html>
{{ end }}
```

MAIN.CSS

```
/* FontAwesome */
@import url("font-awesome.min.css");
@font-face{font-family:'FontAwesome';
        src:url('../fonts/fontawesome-webfont.eot?v=4.6.3');
        src:url('../fonts/fontawesome-webfont.eot?#iefix&v=4.6.3') format('embedded-opentype'),
        url('../fonts/fontawesome-webfont.woff2?v=4.6.3') format('woff2'),
        url('../fonts/fontawesome-webfont.woff?v=4.6.3') format('woff'),
        url('../fonts/fontawesome-webfont.ttf?v=4.6.3') format('truetype'),
        url('../fonts/fontawesome-webfont.svg?v=4.6.3#fontawesomeregular') format('svg');
        font-weight:normal;font-style:normal}.fa{display:inline-block;
        font:normal normal normal 14px/1 FontAwesome;
        font-size:inherit;text-rendering:auto;
        -webkit-font-smoothing:antialiased;
        -moz-osx-font-smoothing:grayscale}

/* Reset/Serialization */

        html, body, div, span, applet, object, iframe, h1, h2, h3, h4, h5, h6, p, blockquote, pre, a, abbr, acronym,
        address, big, cite, code, del, dfn, em, img, ins, kbd, q, s, samp, small, strike, strong, sub, sup, tt, var, b,
        u, i, center, dl, dt, dd, ol, ul, li, fieldset, form, label, legend, table, caption, tbody, tfoot, thead, tr, th,
        td, article, aside, canvas, details, embed, figure, figcaption, footer, header, hgroup, menu, nav, output, ruby,
        section, summary, time, mark, audio, video {
                margin: 0;
                padding: 0;
                border: 0;
                font: normal 100% inherit;
                vertical-align: baseline;
        }
        article, aside, details, figcaption, figure, footer, header, hgroup, menu, nav, section {
                display: block;
        }
        body {
                line-height: 1;
        }
        ol, ul {
                list-style: none;
        }
        blockquote, q {
                quotes: none;
        }
        blockquote:before, blockquote:after, q:before, q:after {
                content: none;
        }
        table {
                border-collapse: collapse;
                border-spacing: 0;
        }
        body {
     -webkit-text-size-adjust: none;
   }

/* Box Model */

        *, *:before, *:after {
                -moz-box-sizing: border-box;
                -webkit-box-sizing: border-box;
                box-sizing: border-box;
        }

/* Containers, Screen */

        .container {
                margin-left: auto;
                margin-right: auto;
        }
        .container {
                width: 1400px;
        }
        @media screen and (min-width: 961px) and (max-width: 1880px) {
```

```
			.container {
				width: 1200px;
			}
		}
		@media screen and (min-width: 961px) and (max-width: 1620px) {
			.container {
				width: 960px;
			}
		}
		@media screen and (min-width: 961px) and (max-width: 1320px) {
			.container {
				width: 100%;
			}
		}
		@media screen and (max-width: 960px) {
			.container {
				width: 100%;
			}
		}
		@media screen and (max-width: 736px) {
			.container {
				width: 100% !important;
			}
		}

/* Grid */

		.row {
			border-bottom: solid 1px transparent;
			-moz-box-sizing: border-box;
			-webkit-box-sizing: border-box;
			box-sizing: border-box;
		}
		.row > * {
			float: left;
			-moz-box-sizing: border-box;
			-webkit-box-sizing: border-box;
			box-sizing: border-box;
		}
		.row:after, .row:before {
			content: '';
			display: block;
			clear: both;
			height: 0;
		}
		.row.uniform > * > :first-child {
			margin-top: 0;
		}
		.row.uniform > * > :last-child {
			margin-bottom: 0;
		}

/* Basic */

		body {
			background: #fff;
			font-family: 'Source Sans Pro', sans-serif;
			font-size: 19pt;
			font-weight: 300;
			line-height: 1.75em;
			color: #888;
		}
		body.is-loading * {
			-moz-transition: none !important;
			-webkit-transition: none !important;
			transition: none !important;
			-moz-animation: none !important;
			-webkit-animation: none !important;
			animation: none !important;
		}
		input, textarea, select {
			font-family: 'Source Sans Pro', sans-serif;
			font-size: 18pt;
			font-weight: 300;
			line-height: 1.75em;
			color: #888;
```

```
}
h1, h2, h3, h4, h5, h6 {
		font-weight: 300;
		color: #666;
		line-height: 1.5em;
}
h1 a h2 a, h3 a, h4 a, h5 a, h6 a {
		color: inherit;
		text-decoration: none;
}
h1 a strong, h2 a strong, h3 a strong, h4 a strong, h5 a strong, h6 a strong {
		color: #333;
}
h2 {
		font-size: 1.8em;
		letter-spacing: -1px;
}
h2.alt {
		color: #888;
}
h2.alt strong {
		color: #666;
}
h3 {
		font-size: 1.3em;
}
header {
		margin: 0 0 2em 0;
}
header > p {
		 margin: 1em 0 0 0;
 }
footer {
		margin: 2em 0 0 0;
}
strong, b {
		font-weight: 300;
		color: #666;
}
em, i {
		font-style: italic;
}
a {
		text-decoration: none;
		color: inherit;
		border-bottom: dotted 1px rgba(128, 128, 128, 0.5);
		-moz-transition: color 0.35s ease-in-out, border-bottom-color 0.35s ease-in-out;
		-webkit-transition: color 0.35s ease-in-out, border-bottom-color 0.35s ease-in-out;
		transition: color 0.35s ease-in-out, border-bottom-color 0.35s ease-in-out;
		outline: 0;
}
a:hover {
		color: #E27689;
		border-bottom-color: rgba(255, 255, 255, 0);
}
sub {
		position: relative;
		top: 0.5em;
		font-size: 0.8em;
}
sup {
		position: relative;
		top: -0.5em;
		font-size: 0.8em;
}
hr {
		border: 0;
		border-top: solid 1px #ddd;
}
blockquote {
		border-left: solid 0.5em #ddd;
		padding: 1em 0 1em 2em;
		font-style: italic;
}
p, ul, ol, dl, table {
		margin-bottom: 2em;
		}
		br.clear {
			clear: both;
		}

/* Sections/Article */

		section, article {
			margin-bottom: 3em;
			height: 100vh; /* todo */
		}
		section > :last-child,
		section > .container, section:last-child, article > :last-child,
		article > .container, article:last-child {
			margin-bottom: 0;
		}
		.row > section, .row > article {
			margin-bottom: 0;
		}
  .container article{
    height: 60vh;
    overflow: auto;
  }

/* Image */

		.image {
			display: inline-block;
			border: 0;
		}
		.image img {
			display: block;
			width: 100%;
		}
		.image.avatar48 {
			width: 48px;
			height: 48px;
			background: #1C2022;
		}
		.image.avatar48 img {
			width: 48px;
			height: 48px;
		}
		.image.fit {
			display: block;
			width: 100%;
		}
		.image.featured {
			display: block;
			width: 100%;
			margin: 0 0 2em 0;
		}
		.image.left {
			float: left;
			margin: 0 2em 2em 0;
		}
		.image.centered {
			display: block;
			margin: 0 0 2em 0;
		}
		.image.centered img {
			margin: 0 auto;
			width: auto;
		}

/* List */

		ul.default {
			list-style: disc;
			padding-left: 1em;
		}
		ul.default li {
			padding-left: 0.5em;
		}
		ul.icons {
			cursor: default;
		}
		ul.icons li {
			display: inline-block;
		}
		ul.icons a {
			display: inline-block;
			width: 2em;
			height: 2em;
			line-height: 2em;
			text-align: center;
			border: 0;
		}
		ol.default {
			list-style: decimal;
			padding-left: 1.25em;
		}
		ol.default li {
			padding-left: 0.25em;
		}

/* Form */

		form label {
			display: block;
			text-align: left;
			margin-bottom: 0.5em;
		}
		form input[type="text"],
		form input[type="search"],
		form input[type="time"],
		form input[type="password"],
		form input[type="number"],
		form select,
		form textarea {
			position: relative;
			-webkit-appearance: none;
			display: block;
			outline: 0;
			background: #fff;
			background: rgba(255, 255, 255, 0.75);
			width: 80%;
			border-radius: 0.35em;
			padding: 0.5em 0.6em 0.5em 0.6em;
			box-shadow: inset 0 0.1em 0.1em 0 rgba(0, 0, 0, 0.05);
			border: solid 1px rgba(0, 0, 0, 0.15);
			-moz-transition: all 0.35s ease-in-out;
			-webkit-transition: all 0.35s ease-in-out;
			transition: all 0.35s ease-in-out;
			margin: 5px;
		}
		form input[type="text"]:focus,
		form input[type="search"]:focus,
		form input[type="time"]:focus,
		form input[type="password"]:focus,
		form input[type="number"]:focus,
		form select:focus,
		form textarea:focus {
			box-shadow: 0 0 2px 1px #8ebebc;
			background: #fff;
		}
		form input[type="text"],
		form input[type="search"],
		form input[type="time"],
		form input[type="password"],
		form input[type="number"],
		form select {
			line-height: 1em;
		}
		form textarea {
			min-height: 10em;
		}
		form .formerize-placeholder {
			color: #555 !important;
		}
		form ::-webkit-input-placeholder {
			color: #555 !important;
		}
		form :-moz-placeholder {
			color: #555 !important;
		}
		form ::-moz-placeholder {
			color: #555 !important;
		}
		form :-ms-input-placeholder {
			color: #555 !important;
		}
		form ::-moz-focus-inner {
			border: 0;
		}
		#logout{
			position: fixed;
			top: 0.8em;
			right: 0.8em;
			width: 1.8em;
			height: 1.8em;
			z-index: 996;
			cursor: pointer;
		}
		#sim{
			position: fixed;
			top: 0.7em;
			right: 4em;
			width: 2.2em;
			height: 2em;
			z-index: 996;
			cursor: pointer;
		}
		#titleIpt{
			width: 10em;
			display: inline-block;
		}
		#user form{
			margin-bottom: 3em;
		}
		form input[type="number"]{
			width: 30%;
		}
  form input:read-only {
    background: #EEE;
  }
		.smaller input[type="time"],.smaller input[type="number"]{
			width: 6em; !important;
			display: inline-block; !important;
		}

/* Table */

		table {
			width: 100%;
		}
		table.default {
			width: 100%;
			text-align: left;
		}
		table.default td {
			padding: 0.5em 1em 0.5em 1em;
		}
		table.default th {
			text-align: left;
			padding: 0.5em 1em 0.5em 1em;
			color: #fff;
			background: #222729;
		}
		table.default thead {
			background: #444;
			color: #fff;
		}
		table.default tfoot {
			background: #eee;
		}
		#eOffset{
			width: 4em;
		}
		#stateTable{
			padding: 0;
			margin: 0;
		}

/* Button */

		input[type="button"],
		input[type="submit"],
		input[type="reset"],
		button,
		.button {
			position: relative;
			display: inline-block;
			border-radius: 0.35em;
			color: #fff !important;
			text-decoration: none;
			padding: 0.3em 1em 0.3em 1em;
			background-color: #8ebebc;
			border: 0;
			margin: 5px;
			cursor: pointer;
			background-image: -moz-linear-gradient(top, rgba(0, 0, 0, 0), rgba(0, 0, 0, 0.15));
			background-image: -webkit-linear-gradient(top, rgba(0, 0, 0, 0), rgba(0, 0, 0, 0.15));
			background-image: -ms-linear-gradient(top, rgba(0, 0, 0, 0), rgba(0, 0, 0, 0.15));
			background-image: linear-gradient(top, rgba(0, 0, 0, 0), rgba(0, 0, 0, 0.15));
			-moz-transition: background-color 0.35s ease-in-out;
			-webkit-transition: background-color 0.35s ease-in-out;
			transition: background-color 0.35s ease-in-out;
		}
		input[type="button"]:hover,
		input[type="submit"]:hover,
		input[type="reset"]:hover,
		button:hover,
		.button:hover {
			background-color: #9ececc;
		}
		input[type="button"]:active,
		input[type="submit"]:active,
		input[type="reset"]:active,
		button:active,
		.button:active {
			background-color: #7eaeac;
		}
  .addItem{
    width: 1.2em;
    height: 1.2em;
    padding: 0;
    font-size: larger;
  }
		.delete-btn{
			width: 1em;
			height: 1em;
			opacity: 0.8;
			cursor: pointer;
		}
		.edit-btn{
			width: 1em;
			height: 1em;
			opacity: 0.8;
			cursor: pointer;
		}
		input[type="submit"]:disabled {
			cursor: not-allowed;
			color: dimgray;
			background-color: lightgray;
		}

/* Item */

		.item {
			box-shadow: 0 0.05em 0.15em 0 rgba(0, 0, 0, 0.05);
			margin-bottom: 40px;
		}
		.item header {
```

```
                        background: #fff;
                        margin: 0;
                        padding: 1em 0 1em 0;
                        font-size: 0.8em;
                }
                .item header h3 {
                        font-size: 1em;
                }
                .oItem{
                        height: 5em;
                        font-size: smaller;
                float: left;
    margin: 1em;
                        box-shadow: 0 0.3em 0.3em 0 rgba(80, 80, 80, 0.5);
                }
                .Switch{
                        cursor: pointer;
    word-wrap: break-word;
                        width: 6em;
                        height: 5em;
                }
                .Range{
                        width: 4em;
                        height: 1.2em;
                }
                .Number{
                        width: 4em;
                        height: 1.2em;
                }


/* Icons */

                .icon {
                        text-decoration: none;
                }
                .icon:before {
                        display: inline-block;
                        font-family: FontAwesome;
                        font-size: 1.4em;
                        text-decoration: none;
                        font-style: normal;
                        font-weight: normal;
                        line-height: 1;
                        -webkit-font-smoothing: antialiased;
                        -moz-osx-font-smoothing: grayscale;
                }
                .icon > .label {
                        display: none;
                }

/* Header */

                #header {
                        display: -moz-flex;
                        display: -webkit-flex;
                        display: -ms-flex;
                        display: flex;
                        -moz-flex-direction: column;
                        -webkit-flex-direction: column;
                        flex-direction: column;
                        -moz-justify-content: space-between;
                        -webkit-justify-content: space-between;
                        justify-content: space-between;
                        background: #222629;
                        box-shadow: inset -0.25em 0 0.25em 0 rgba(0, 0, 0, 0.1);
                        color: #fff;
                        height: 100%;
                        left: 0;
                        overflow-y: auto;
                        position: fixed;
                        text-align: right;
                        top: 0;
                        width: 350px;
                }
                #header .top {
```

```
		-moz-flex-grow: 1;
		-webkit-flex-grow: 1;
		flex-grow: 1;
}
#header .bottom {
		-moz-flex-shrink: 0;
		-webkit-flex-shrink: 0;
		flex-shrink: 0;
		padding: 1.5em 0 0.75em 0;
}
#header .bottom > :last-child {
		margin-bottom: 0;
}
#header .icons {
		font-size: 1.8em;
		text-align: center;
}
#header .icons a {
		color: #41484c;
		-moz-transition: color 0.35s ease-in-out;
		-webkit-transition: color 0.35s ease-in-out;
		transition: color 0.35s ease-in-out;
}
#header .icons a:hover {
		color: #fff;
}
#logo {
		position: relative;
		margin: 2em 1.5em 1.5em 40%;
		min-height: 48px;
		cursor: default;
}
#logo h1 {
		position: relative;
		color: #fff;
		font-weight: 600;
		font-size: 1em;
		line-height: 1em;
		margin: auto;
}
#logo p {
		position: relative;
		display: block;
		font-size: 0.5em;
		color: rgba(255, 255, 255, 0.5);
		line-height: 1.25em;
		margin: auto;
}
#logo .image {
		position: absolute;
		left: 0;
		top: 0;
}
#nav ul {
		margin-bottom: 0;
}
#nav ul li a {
		display: block;
		padding: 0.6em 2.5em 0.6em 2.5em;
		color: rgba(255, 255, 255, 0.5);
		text-decoration: none;
		outline: 0;
		border: 0;
		-moz-transition: none;
		-webkit-transition: none;
		transition: none;
}
#nav ul li a span {
		position: relative;
		display: block;
		font-size: 1.1em;
}
#nav ul li a span:before {
		position: absolute;
		left: 0;
		color: #41484c;
```

```
                        text-align: center;
                        width: 1.25em;
                        line-height: 1em;
  margin-left: -0.5em;
            }
            #nav ul li a.active {
                        background: rgba(0, 0, 0, 0.15);
                        box-shadow: inset 0 0 0.25em 0 rgba(0, 0, 0, 0.125);
                        color: #fff;
            }
            #nav ul li a.active span:before {
                        color: #e27689;
            }
            #message{
                        display: none;
                        position: fixed;
                        top: 0; !important;
                        margin-left: 30%;
                        width: 40%;
                        max-width: 15em;
                        text-align: center;
                        background-color: darkred;
                        color: #dddddd;
                        font-family: FontAwesome;
                        text-decoration: none;
                        font-style: italic;
                        font-weight: normal;
                        font-size: larger;
                        padding: 0.3em;
                        -webkit-box-shadow: 0px 2px 10px 0px rgba(0,0,0,0.3);
                        -moz-box-shadow: 0px 2px 10px 0px rgba(0,0,0,0.3);
                        box-shadow: 0px 2px 10px 0px rgba(0,0,0,0.3);
                        z-index: 999;
            }

/* Add */
 .add{
   display: none;
   z-index: 996;
 }
 .addLabel{
   display: block;
   background-image: url('../../images/add.png');
   background-size: 100%;
   width: 3em;
   height: 3em;
   position: fixed;
   bottom: 1em;
   right: 1em;
   text-align: center;
   cursor: pointer;
   z-index: 996;
 }
 .add[type=checkbox]:checked + label +  #addMenu{
   display: block;
 }
 #addMenu{
   display: none;
   position: fixed;
   width: 100px;
   right: 2em;
   bottom: 3.5em;
                        z-index: 996;
 }
 #addMenu ul{
   margin: 0.8em;
   padding: 0;
   list-style: none;
   line-height: 1.1em;
   display: block;
 }
 #addMenu li{
   margin: 0.3em 0em;
   display: inline-block;
   padding: 0 1em 0 1em;
 }
```

```
    #addMenu a{
        text-decoration: none;
        font-family: 'Source Sans Pro', sans-serif;
        font-size: larger;
        color: #0d1217;
    }
    #addMenu li:hover{
        background-color: rgba(160,40,30,0.6);
        -webkit-box-shadow: 0 0 0 0 rgba(0,0,0,0.45);
        -moz-box-shadow: 0 0 0 0 rgba(0,0,0,0.45);
        box-shadow: 0 0 0 0 rgba(0,0,0,0.45);
    }
    #addMenu li:active{
        background-color: rgba(160,40,30,0.85);
        -webkit-box-shadow: 0 0 0 0 rgba(0,0,0,0.45);
        -moz-box-shadow: 0 0 0 0 rgba(0,0,0,0.45);
        box-shadow: 0 0 0 0 rgba(0,0,0,0.45);
    }

/* Dialog */

        .dialog {
                z-index: 997;
                position: absolute;
                display: none;
                width: 66vw;
                margin-left: 24vw;
                overflow: hidden;
                padding: 0;
                border: 0;
                border-radius: 5px;
                box-shadow: 0 3px 33px 0 rgba(0, 0, 0, .3);
        }
        .dialog-header {
                padding: 0.3em 0.5em;
                background-color: #1C2022;
                margin-bottom: 0.4em;
        }
        .dialog-content {
                padding: 0.8em 0.8em;
                color: #bfbfbf;
                background-color: #fff;

        }
        .dialog-content th{
                font-size: larger;
        }
        .btn-group {
                padding: 0.1em 0.2em;
                text-align: right;
        }
        .btn {
                padding: 0.1em 0.2em;
                cursor: pointer;
                color: darkgray;
                border: 1px solid;
                border-radius: 3px;
                background-color: gray;
                float: right;
        }
        .btn:disabled {
                 cursor: not-allowed;
                 color: dimgray;
                 background-color: lightgray;
         }
        .btn:disabled:hover {
                color: dimgray;
                background-color: lightgray;
        }
        .btn-add:hover {
                 background-color: lightgray;
        }
        .btn-cancel:hover {
                background-color: lightgray;
        }
        #addType{
```

```
                        top: 306vh;
                }
                #addDevice{
                        top: 106vh;
                }
                #addEvent{
                        top: 206vh;
                }
   .fromTo{
      float: left;
      display: inline;
      width: 25%;
      -webkit-appearance: none;
      outline: 0;
      background: #fff;
      background: rgba(255, 255, 255, 0.75);
      border-radius: 0.35em;
      padding: 0.5em 0.6em 0.5em 0.6em;
      box-shadow: inset 0 0.1em 0.1em 0 rgba(0, 0, 0, 0.05);
      border: solid 1px rgba(0, 0, 0, 0.15);
      -moz-transition: all 0.35s ease-in-out;
      -webkit-transition: all 0.35s ease-in-out;
      transition: all 0.35s ease-in-out;
      margin: 5px;
   }
   .centered{
      margin-left: 10%;
   }
   .item td{
      padding: 0.5em;
   }
   .item td select{
                        padding: 0.6em 0.6em 0.6em 0.6em;
      width: 60%;
      float: left;
   }
   .addItem{
      float: right;
   }
   .hide{
      display: none;
   }


/* Footer */

                #footer {
                        margin-left: 350px;
                        text-align: center;
                        background-color: #dce3e2;
                        padding: 3em 0 4em 0;
                        box-shadow: inset 0 1px 0 0 rgba(0, 0, 0, 0.05), inset 0 0.1em 0.1em 0 rgba(0, 0, 0, 0.025);
                        font-size: 0.8em;
                }
                #footer .copyright {
                        cursor: default;
                        margin: 0;
                }
                #footer .copyright li {
                        display: inline-block;
                        line-height: 1em;
                        border-left: solid 1px rgba(128, 128, 128, 0.35);
                        padding: 0 0 0 0.5em;
                        margin: 0 0 0 0.5em;
                }
                #footer .copyright li:first-child {
                        border-left: 0;
                        padding-left: 0;
                        margin-left: 0;
                }

/* Main */

                #main {
                        margin-left: 350px;
                }
```

```
		#main > section {
			margin: 0;
			overflow: hidden;
			padding: 4em 0;
			box-shadow: inset 0 1px 0 0 rgba(0, 0, 0, 0.05), inset 0 0.1em 0.1em 0 rgba(0, 0, 0, 0.025);
			text-align: center;
			background-color: #222629;
		}
		#main > section.dark {
			color: #ddd;
			color: rgba(255, 255, 255, 0.75);
		}
		#main > section.dark h2, #main > section.dark h3, #main > section.dark h4,
		#main > section.dark h5, #main > section.dark h6 {
			color: inherit;
		}
		#main > section.dark strong {
			color: #fff;
			border-color: inherit;
		}
		#main > section.dark a {
			color: #fff;
			border-color: inherit;
		}
		#main > section.dark a:hover {
			border-bottom-color: rgba(255, 255, 255, 0);
		}
		#main > section.cover {
			padding: 6em 0;
			background-size: cover;
			background-position: center center;
		}
		#main > section.one {
			top: 0; !important;
			background: #81918E url("../../images/banner.jpg");
		}
		#main > section.two {
			background-color: #f5fafa;
		}
		#main > section.three {
			background-color: #ecf1f1;
		}
		#main > section.four {
			background-color: #e8edec;
		}
		#simUi h3{
			font-style: italic;
			text-decoration: none;
			font-family: FontAwesome;
			font-style: italic;
			font-weight: normal;
		}
		#simUi header{
			padding-bottom: 0.1em;
		}

/*** Responsive ***/
/* Wide */

		@media screen and (min-width: 961px) and (max-width: 1880px) {
			/* Basic */
			body, input, textarea, select {
				font-size: 17pt;
			}
			/* Header */
			#header {
				width: 250px;
			}
			/* Footer */
			#footer {
				margin-left: 250px;
			}
			/* Main */
			#main {
				margin-left: 250px;
			}
		}

/* Normal */

		@media screen and (min-width: 961px) and (max-width: 1620px) {
			/* Main */
			#main > section {
				padding: 3em 0;
			}
			#main section.cover {
				padding: 5em 0;
			}
		}

/* Small */

		@media screen and (min-width: 961px) and (max-width: 1320px) {
			/* Basic */
			body, input, textarea, select {
				font-size: 16pt;
			}
			.container {
				padding: 0 2em 0 2em;
			}
			/* List */
			ul.icons li a {
				width: 1.75em;
			}
			/* Item */
			.item {
				margin-bottom: 20px;
			}
			/* Header */
			#header {
				width: 16%;
			}
			#logo h1 {
				margin-right: 30%;
			}
			#logo p {
				margin-right: 30%;
			}
			#logo .image {
				position: relative;
				margin: 0 1em 0.5em 0;
			}
			#nav ul li a {
				font-size: 0.8em;
				padding-top: 0.5em;
				padding-bottom: 0.2em;
    padding-left: 1.6em;
			}
			#nav ul li a span {
				padding-right: 1.5em;
    text-align: left;
			}
			#nav ul li a span:before {
				left: 100%;
				margin-left: -0.4em;
				line-height: 1.25em;
			}
			/* Footer */
			#footer {
				margin-left: 16%;
			}
			/* Main */
			#main {
				margin-left: 16%;
			}
  /* Add */
  #addMenu{
    right: 1.1em;
  }
		}

/* Smaller */
```

```
#headerToggle {
		display: none;
}
@media screen and (max-width: 960px) {
		/* Basic */
		html, body {
				overflow-x: hidden;
		}
		body, input, textarea, select {
				font-size: 16pt;
		}
		header br {
				display: none;
		}
		.container {
				padding: 0 2em 0 2em;
		}
		/* Item */
		.item {
				margin-bottom: 15px;
		}
		/* List */
		ul.icons a {
				width: 1.75em;
				font-size: 1.25em;
		}
		/* Header */
		#header {
				-moz-backface-visibility: hidden;
				-webkit-backface-visibility: hidden;
				backface-visibility: hidden;
				-moz-transform: translateX(-275px);
				-webkit-transform: translateX(-275px);
				-ms-transform: translateX(-275px);
				transform: translateX(-275px);
				-moz-transition: -moz-transform 0.5s ease;
				-webkit-transition: -webkit-transform 0.5s ease;
				transition: transform 0.5s ease;
				-webkit-overflow-scrolling: touch;
				display: block;
				height: 100%;
				left: 0;
				overflow-y: auto;
				position: fixed;
				top: 0;
				width: 250px;
				z-index: 10002;
				background: #222729;
				box-shadow: inset -0.25em 0 0.25em 0 rgba(0, 0, 0, 0.125);
		}
		#header .top {
				position: relative;
		}
		#header .bottom {
				border-top: solid 1px rgba(255, 255, 255, 0.05);
				box-shadow: 0 -1px 0 0 rgba(0, 0, 0, 0.15);
				padding-top: 2em;
				margin-top: 2em;
				position: relative;
		}
		#logo {
				margin: 1.5em 1.25em 1.25em 1.25em;
		}
		#nav ul li a {
				padding: 0.2em 1.25em 0.2em 1.25em;
		}
		#headerToggle {
				-moz-backface-visibility: hidden;
				-webkit-backface-visibility: hidden;
				backface-visibility: hidden;
				-moz-transition: -moz-transform 0.5s ease;
				-webkit-transition: -webkit-transform 0.5s ease;
				transition: transform 0.5s ease;
				display: block;
				height: 2.25em;
		left: 0;
		position: fixed;
		top: 0;
		width: 3.25em;
		z-index: 10001;
	}
	#headerToggle .toggle {
		position: absolute;
		left: 0;
		top: 0;
		width: 100%;
		height: 100%;
		outline: 0;
		border: 0;
	}
	#headerToggle .toggle:before {
		font-family: FontAwesome;
		text-decoration: none;
		font-style: normal;
		font-weight: normal;
		-webkit-font-smoothing: antialiased;
		-moz-osx-font-smoothing: grayscale;
		content: '\f0c9';
		color: #fff;
		font-size: 18px;
		line-height: 2.25em;
		background: rgba(128, 136, 144, 0.5);
		border-radius: 0.35em;
		text-align: center;
		position: absolute;
		left: 0.5em;
		top: 0.5em;
		display: block;
		width: 3.25em;
		height: 2.25em;
	}
	body.header-visible #main {
		-moz-transform: translateX(250px);
		-webkit-transform: translateX(250px);
		-ms-transform: translateX(250px);
		transform: translateX(250px);
	}
	body.header-visible #headerToggle {
		-moz-transform: translateX(250px);
		-webkit-transform: translateX(250px);
		-ms-transform: translateX(250px);
		transform: translateX(250px);
	}
	body.header-visible #header {
		-moz-transform: translateX(0);
		-webkit-transform: translateX(0);
		-ms-transform: translateX(0);
		transform: translateX(0);
	}
	/* Footer */
	#footer {
		margin-left: 0;
	}
	/* Main */
	#main {
		-moz-backface-visibility: hidden;
		-webkit-backface-visibility: hidden;
		backface-visibility: hidden;
		-moz-transition: -moz-transform 0.5s ease;
		-webkit-transition: -webkit-transform 0.5s ease;
		transition: transform 0.5s ease;
		padding-bottom: 1px;
		margin-left: 0;
	}
	#main > section {
		padding: 3em 0;
	}
	#main section.cover {
		padding: 4em 0;
	}
/* Add */
```

```
    #addMenu{
       right: 1.1em;
    }
               }

/* Mobile */
               @media screen and (max-width: 736px) {
                              /* Basic */
                              body, input, textarea, select {
                                             font-size: 14pt;
                              }
                              h2 {
                                             font-size: 1.5em;
                                             letter-spacing: 0;
                                             font-weight: 300;
                              }
                              .container {
                                             padding: 0 15px 0 15px;
                              }
                              /* List */
                              ul.icons a {
                                             width: 2em;
                                             font-size: 1.25em;
                              }
                              /* Main */
                              #main > section {
                                             padding: 2em 0;
                              }

                              #main section.cover {
                                             padding: 4em 0em;
                              }

                              #main section.cover header {
                                             padding: 0 1em;
                              }
                              /* Footer */
                              #footer .copyright li {
                                             display: block;
                                             line-height: 1.25em;
                                             border: 0;
                                             padding: 0;
                                             margin: 1em 0 0 0;
                              }
                              #footer .copyright li:first-child {
                                             margin-top: 0;
                              }
    /* Add */
    #addMenu{
       right: 0.9em;
    }
               }
```

MAIN.JS

```
"use strict";
//////////////////////////////////////////////////////////////////////////////////////
////// All
//////////////////////////////////////////////////////////////////////////////////////

// Init
window.addEventListener("load", function(){ init() });

// Helper
let ajaxIntervalOverview = 3000;
let ajaxIntervalState = 1000;
let address = "http://localhost:4242";

function $id(id){
   return document.getElementById(id);
}
function $tag(tag){
   return document.getElementsByTagName(tag);
}
function $name(name){
   return document.getElementsByName(name);
}

// Start function
function init() {
   // Serialization
   serialization();
   // Set timeout for overview if ui is open
   if ($id("home")) {
      // Get device state and write at start/interval
      getDevices("once");
      getDevices();
   }
   // Get states for simulator if sim is open
   if ($id("simUi")) {
      // Get sim states and write at start/interval
      getSimStates("once");
      getSimStates();
   }
   // EventHandler
   loadEventListeners();
   // Show Message
   showMessage();
}

// Function to initialise event listeners
function loadEventListeners() {
   // Add event listener just on interface site
   // if items exist
   if ($id("addMenu")) {
      dialogHandlers();
      actionButtonHandlers();
   }
   // Add event listener to simulater ui
   // if items exist
   if ($id("simUi")) {
      actionButtonHandlersSim();
   }
   // Toggle menu
   $id("headerToggle").addEventListener("click", ()=> showNav())
}

// Function to decode names and ids
// Params: text -> text to be converted
// Return: converted text
function decodeHTMLEntities(text) {
   let entities = [
      ['amp', '&'], ['apos', '\''], ['#x27', '\''],
      ['#x2F', '/'], ['#39', '\''], ['#47', '/'],
      ['lt', '<'], ['gt', '>'], ['nbsp', ' '], ['quot', '"']
   ];
   for (let i = 0, max = entities.length; i < max; ++i)
      text = text.replace(new RegExp('&'+entities[i][0]+';', 'g'), entities[i][1]);
   return text;
}

////////////////////////////////////////////////////////////////////////////////////
///// Interface
////////////////////////////////////////////////////////////////////////////////////

// Function to initialise the listeners for the dialog windows
function dialogHandlers() {
   // helper function
   function action(bId, iName, dId, show, read) {
      $name(iName)[0].readOnly = false; // Input
      $id(bId).value = "Add"; // Button
      // Dialog
      $id(dId).style.display = show ? "block" : "none";
      updateDialogTitle(dId, true);
   }
   // to show the dialogs
   $id("addDeviceBtn").addEventListener("click", ()=>{
      action("addD","dName","addDevice", true);
   });
   $id("addTypeBtn").addEventListener("click", ()=>{
      action("addT","tName","addType", true);
```

```
    });
    $id("addEventBtn").addEventListener("click", ()=>{
        action("addE","eName","addEvent", true);
    });
    // to hide the dialogs
    $id("cancelD").addEventListener("click", ()=>{
        action("addD","dName","addDevice", false);
    });
    $id("cancelT").addEventListener("click", ()=>{
        action("addT","tName","addType", false);
    });
    $id("cancelE").addEventListener("click", ()=>{
        action("addE","eName","addEvent", false);
    });
    // Listener to add/cancel Item to Event Dialog
    $id("eDevices-btn").addEventListener("click", ()=> {
        addItemEvent();
    });
    $id("cancelE").addEventListener("click", ()=> {
        cancelItemEvent();
    });
    // Listener to update type selection
    $id("tKind").addEventListener("change", ()=>{
        changeMinMax();
    });

    // Check all items set from user to enable add button
    // Just if all inputs are set, something can be added
    let buttons = ["addE", "addEvent", "addD", "addDevice", "addT", "addType"];
    for(let i = 1; i < buttons.length; i=i+2) {
        let elements = $id(buttons[i]).getElementsByTagName("input");
        // Check changes input fields
        for(let e = 0; e < elements.length; e++) {
            elements[e].addEventListener("change", () => {
                let value = true;
                for (let e2 = 0; e2 < elements.length; e2++) {
                    if (elements[e2].value == "") {
                        value = false;
                    }
                }
                $id(buttons[i-1]).disabled = !value;
            });
        }
        // Check at hover dialog
        $id(buttons[i]).addEventListener("mouseover", ()=>{
            for(let e = 0; e < elements.length; e++) {
                let value = true;
                for (let e2 = 0; e2 < elements.length; e2++) {
                    if (elements[e2].value == "") {
                        value = false;
                    }
                }
                $id(buttons[i-1]).disabled = !value;
            }
        });
    }
}

// Function optimize the template
function serialization() {
    let body = $tag('body')[0];
    // Disable transitions until the page has loaded
    body.className += 'is-loading';
    window.addEventListener('load', function () {
        body.className -= 'is-loading';
    });
}

// If there is a error message, it will be shown for short time
function showMessage(){
    let text = $id("message").childNodes[1].textContent;
    if(text.substr(0,5) !== "Error"){
        $id("message").style.backgroundColor = "darkgreen";
    }
    if(text !== "") {
        $id("message").style.display = "block";
        setTimeout( ()=>{
            $id("message").style.display = "none";
            text = ""
        }, 3000)
    }
}

// Funktion to add an Item to Event Dialog for multiple devices
function addItemEvent() {
    let pattern = $id("eDevices1").cloneNode(true);
    $id("eDevices-table").getElementsByTagName('tbody')[0].appendChild(pattern);
}

// Funktion to remove Items from Event Dialog
function cancelItemEvent() {
    let pattern = $id("eDevices1").cloneNode(true);
    let elements = $name("eDevices-tr");
    let table = $id("eDevices-table").getElementsByTagName('tbody')[0];
    for(let i = 0; i < elements.length; i++) {
        table.removeChild(elements[i]);
    }
    table.appendChild(pattern);
}

// Funktion to change Min Max option to Type selection in dialog
function changeMinMax() {
    let check = false;
    let parent = $id("tDevices-table");
    // Disable for switch (Just on/off)
    if($id("tKind").value != "Switch"){
        if(parent.lastChild.textContent != "Max: "){
            // Create Min/Max inputs
            check = true;
            let trMin = document.createElement("tr");
            let trMax = document.createElement("tr");
            trMin.appendChild(document.createTextNode("Min: "));
            trMax.appendChild(document.createTextNode("Max: "));
            let input = document.createElement("input");
            input.type = "number";
            input.name = "tMin";
            input.value = "0";
            trMin.appendChild(input.cloneNode());
            input.name = "tMax";
            input.value= "1";
            trMax.appendChild(input);
            parent.appendChild(trMin);
            parent.appendChild(trMax);
        }
    }else{
        if(parent.lastChild.textContent == "Max: "){
            // Remove Min/Max inputs
            parent.removeChild(parent.lastChild);
            parent.removeChild(parent.lastChild);
        }
    }
    return check;
}

// Funktion to load listeners to edit/remove images
function actionButtonHandlers() {
    let getArrayFromName = function(tagname) {
        // get the NodeList and transform it into an array
        return Array.prototype.slice.call(document.getElementsByName(tagname));
    };
    // Devices
    let ed = getArrayFromName("DeviceItemEdit");
    let del = getArrayFromName("DeviceItemDel");
    // Events and Types
    let e = ed.concat(getArrayFromName("EventItemEdit"), getArrayFromName("TypeItemEdit"));
    let d = del.concat(getArrayFromName("EventItemDel"), getArrayFromName("TypeItemDel"));
    // Edit
    for(let i = 0; i < e.length; i++){
        e[i].addEventListener("click", ()=>{
            editItem(e[i].id, e[i].name.substr(0,1));
        });
    }
    // Delete
    for(let i = 0; i < d.length; i++){
        d[i].addEventListener("click", ()=>{
            removeItem(d[i].id, d[i].name.substr(0,1));
        });
    }
}

// Function to remove an item
// Send del request to url via ajax
// Params: id -> name to remove, item -> item kind
function removeItem(id, item) {
    let xhr = new XMLHttpRequest();
    let data = address + "/ui/del";
    // Remove from Device list
    switch(item) {
        case "D":
            // Device
            data += "?Item="+item+"&Name="+id.substr(2,id.length);
            xhr.addEventListener("load", function () {
                let parent = $id("devices").getElementsByTagName("table")[0].lastChild;
                parent.removeChild($id(id));
            });
            break;
        case "E":
            // Event
            data += "?Item="+item+"&Name="+id.substr(2,id.length);
            xhr.addEventListener("load", function () {
                let parent = $id("events").getElementsByTagName("table")[0].lastChild;
                parent.removeChild($id(id));
            });
            break;
        case "T":
            // Type
            data += "?Item="+item+"&Name="+id.substr(2,id.length);
            xhr.addEventListener("load", function () {
                let parent = $id("types").getElementsByTagName("table")[0].lastChild;
                parent.removeChild($id(id));
            });
            break;
        default:
    }
    // Remove from overview
    let overview = $id("home").getElementsByTagName("article")[0];
    for(let e in overview.childNodes){
        if(overview.childNodes[e].id == "O_"+id.split("D_")[1]){
            overview.removeChild(overview.childNodes[e]);
        }
    }
    // Remove from server
    xhr.open("GET", data);
    xhr.send();
}

// Function to edit an item (call dialog)
// Params: id -> name to edit, item -> item kind
function editItem(id, item) {
    let element;
    let trs = document.getElementsByTagName("tr");
    for(let e in trs){
        if(trs[e].id == id){
            element = trs[e];
        }
    }
    // Get the current data to dialog window and change button
    // Name is unique and can't be updated. Here a new device has to be create
    switch(item) {
        case "D":
            // Device
            $name("dName")[0].value = element.childNodes[1].textContent;
            $name("dName")[0].readOnly = true;
            $name("dRoom")[0].value = element.childNodes[3].textContent;
            $name("dType")[0].value = element.childNodes[5].textContent;
            // Show
            updateDialogTitle("addDevice", false);
            $id("addD").value = "Update";
            $id("addDevice").style.display = "block";
            break;
        case "E":
            // Event
            $name("eName")[0].value = element.childNodes[1].textContent;
            $name("eName")[0].readOnly = true;
            $name("eTime")[0].value = element.childNodes[3].textContent;
            $name("eOffset")[0].value = element.childNodes[5].textContent;
            let devices = element.childNodes[7].textContent.split(", ");
            for(let i = 0; i < devices.length-1; i++){
                if(i != 0){
                    // new input for devices
                    addItemEvent();
                }
                // Get all devices in inputs for event dialog
                let buffer = devices[i].split("(");
                $name("eDevice")[i].value = buffer[0].split("|")[1] + " | " + buffer[0].split("|")[0];
                //$name("eDevice")[i].readOnly = true;
                $name("to")[i].value = buffer[1].substr(0,buffer[1].length-1);
            }
            // Show
            updateDialogTitle("addEvent", false);
            $id("addE").value = "Update";
            $id("addEvent").style.display = "block";
            break;
        case "T":
            // Type
            $name("tName")[0].value = element.childNodes[1].textContent;
            $name("tName")[0].readOnly = true;
            $name("tKind")[0].value = element.childNodes[3].textContent;
            if(changeMinMax()){
                $name("tMin")[0].value = element.childNodes[5].textContent;
                $name("tMax")[0].value = element.childNodes[7].textContent;
            }
            updateDialogTitle("addType", false);
            $id("addT").value = "Update";
            $id("addType").style.display = "block";
            break;
        default:
    }
}

// Helper to quick modify title of dialog
// Params: id -> Specific dialog, val -> true=Update=>Add, false=Add=>Update
function updateDialogTitle(id , val) {
    let title = $id(id).childNodes[1].childNodes[1].childNodes[1];
    if(val){
        if(title.textContent.split(" ")[0] == "Update") {
            title.textContent = title.textContent.replace("Update", "Add");
        }
    }else{
        if(title.textContent.split(" ")[0] == "Add") {
            title.textContent = title.textContent.replace("Add", "Update");
        }
    }
}

// Function show nav on mobile devices
function showNav() {
    // Toggle
    if($id("header").style.transform == "translateX(-275px)"){
        $id("header").style.transform = "translateX(0px)";
        $id("headerToggle").style.transform = "translateX(+275px)";
    }else{
        $id("header").style.transform = "translateX(-275px)";
        $id("headerToggle").style.transform = "translateX(0px)";
    }
}

// Function to set interval for ajax request of devices
function getDevices(val){
    let getString = "AllDevices";
    let data = "?Get="+getString;
    let url = address + "/ui/get" + data;
    let xhr = new XMLHttpRequest();
    xhr.addEventListener("load", ()=>{
        let response = xhr.responseText;
        if(response != "") {
            buildNewOverview(JSON.parse(response));
        }
    });
    if(val == "once"){
        requestDevices(xhr, url)
    }else{
        setInterval(()=>{requestDevices(xhr, url)}, ajaxIntervalOverview);
    }
}

// Function to get device states updated
function requestDevices(xhr, url){
    xhr.open("GET", url);
    xhr.send();
}

// Function to build a new overwiev with device data
function buildNewOverview(data){
    // Remove old content
    let parent = $id("home").getElementsByTagName("article")[0];
    while(parent.hasChildNodes()){
        parent.removeChild(parent.firstChild);
    }
    // Add new Content
    for(let i = 0; i < data.length; i++){
        let elemtent = createElement(data[i]);
        parent.appendChild(elemtent)
    }
    // Helper function to create elements to set (Type=Kind)
    function createElement(data){
        let item = null;
        let outer = document.createElement("div");
        outer.style.backgroundColor = "#333";
        outer.id = "O_" + data.Name;
        outer.className = "oItem";
        switch (data.Type){
            // Create Switch and click event listener
            case "Switch":
                // Color to show state
                // Red = off, Green = on
                let switchState = function(item, state){
                    if(state == null){
                        item.style.backgroundColor = item.style.backgroundColor == "darkred" ? "darkgreen" : "darkred";
                        item.style.color = item.style.color == "white" ? "black" : "white";
                    }else{
                        if(state == "0"){
                            item.style.backgroundColor = "darkred";
                            item.style.color = "white";
                        }else{
                            item.style.backgroundColor = "darkgreen";
                            item.style.color = "black";
                        }
                    }
                }
                item = document.createElement("div");
                item.addEventListener("click", ()=>{
                    // Actions
                    updateItem(data, item.style.backgroundColor == "darkred" ? "1" : "0");
                    switchState(item);
                });
                item.className += data.Type;
                switchState(item, data.State);
                item.appendChild(document.createTextNode(data.Name));
                item.appendChild(document.createElement("br"));
                item.appendChild(document.createTextNode(data.Room));
                break;
            // Create number and click event listener
            case "Number":
                outer.style.padding = "0.3em";
                item = document.createElement("input");
                item.type = "number";
                item.addEventListener("change", ()=>{
                    updateItem(data, item.value);
                });
                item.className += data.Type;
                item.style.backgroundColor = "grey";
                item.style.color = "black";
                item.value = data.State;
                outer.appendChild(document.createTextNode(data.Name + " - "));
                outer.appendChild(document.createTextNode(data.Room));
                outer.appendChild(document.createElement("br"));
                break;
            // Create Range and click event listener
            case "Range":
                outer.style.padding = "0.3em";
                item = document.createElement("input");
                item.type = "range";
                item.addEventListener("change", ()=>{
                    updateItem(data, item.value);
                });
                item.className += data.Type;
                item.style.backgroundColor = "grey";
                item.style.color = "black";
                item.value = data.State;
                outer.appendChild(document.createTextNode(data.Name + " - "));
                outer.appendChild(document.createTextNode(data.Room));
                outer.appendChild(document.createElement("br"));
                break;
            default:
                item = document.createElement("div");
        }
        outer.appendChild(item);
        return outer;
    }
}

// Function to update new values to db
function updateItem(data, state){
    let get = "?State=" + state + "&Name=" +data.Name;
    let url = address + "/ui/set" + get;
    let xhr = new XMLHttpRequest();
    xhr.open("GET", url);
    xhr.send();
}

//////////////////////////////////////////////////////////////////////////////////////
/// Simulator
//////////////////////////////////////////////////////////////////////////////////////

// Funktion to send action to sim data in ui
function actionButtonHandlersSim() {
    let xhr = new XMLHttpRequest();
    // State
    $id("simStateOn").addEventListener("click", ()=>{
        let url = address + "/sim/set" + "?Set=" + "State" + "&Value=true";
        xhr.open("GET", url);
        xhr.send();
    });
    $id("simStateOff").addEventListener("click", ()=>{
        let url = address + "/sim/set" + "?Set=" + "State" + "&Value=false";
        xhr.open("GET", url);
        xhr.send();
    });
    // Time
    $id("simTime").addEventListener("change", ()=>{
        let url = address + "/sim/set" + "?Set=" + "Time" + "&Value=" + $id("simTime").value;
        xhr.open("GET", url);
        xhr.send();
    });
    // Multiplier
    $id("simMultiplier").addEventListener("change", ()=>{
        let url = address + "/sim/set" + "?Set=" + "Multiplier" + "&Value=" + $id("simMultiplier").value;
        xhr.open("GET", url);
        xhr.send();
    });
    // Check stopped to enable import
    $id("data").addEventListener("mouseover", ()=>{
        if($id("currentState")){
            if($id("currentState").textContent == "On"){
                $id("dataFileSub").disabled = true;
            }else{
                $id("dataFileSub").disabled = false;
            }
        }
    });
}

// Function to set interval for ajax request of devices
function getSimStates(val){
    let getString = "States";
    let data = "?Get="+getString;
    let url = address + "/sim/get" + data;
    let xhr = new XMLHttpRequest();
    xhr.addEventListener("load", ()=>{
        let response = xhr.responseText;
        if(response != "") {
            buildNewStatelist(JSON.parse(response));
        }
    });
    if(val == "once"){
        requestStates(xhr, url)
    }else{
        setInterval(()=>{requestDevices(xhr, url)}, ajaxIntervalState);
    }
}

// Function to get device states updated
function requestStates(xhr, url){
    xhr.open("GET", url);
    xhr.send();
}

// Function to build a new overwiev with device data
function buildNewStatelist(data){
    // Remove old content
    let parent = $id("simUi").getElementsByTagName("header")[0];
    while(parent.hasChildNodes()){
        parent.removeChild(parent.firstChild);
    }
    // Add new Content
    let table = createElement(data);
    parent.appendChild(table);

    // Helper function to create
    // elements to set (Type=Kind)
    function createElement(data){
        let outer = document.createElement("table");
        outer.id = "stateTable";
        outer.style.padding = "0.5em";
        let th = document.createElement("tr");
        for(let i in data){
            if(i != "Id"){
                let td = document.createElement("td");
                td.appendChild(document.createTextNode(i));
                th.appendChild(td);
            }
        }
        outer.appendChild(th);
        let tr = document.createElement("tr");
        for(let i in data){
            if(i != "Id"){
                let td = document.createElement("td");
                let val = data[i];
                if(i == "State"){
                    td.id = "currentState";
                    val = val ? "On" : "Off";
                }
                if(i == "Time"){
                    let h = Math.floor(val/60/60);
                    let m = Math.round((val-h*60*60)/60);
                    val = h + ":" + m;
                }
                // Update action inputs
                if(i == "Multiplier"){
                    $id("simMultiplier").value = data[i];
                }
            td.appendChild(document.createTextNode(val));
            tr.appendChild(td);
        }
      }
      outer.appendChild(tr);
      return outer;
   }
}
```

USER.MODEL.GO

```
// Class for user manipulation
// This class processing data of users (Set data/Delete/Get)
// Package db, login is needed
package models

import (
                "gopkg.in/mgo.v2/bson"
                "net/http"
                "toHuus/db"
                "toHuus/login"
)

// Initialisation
const CookieName = "session"
const DbCollUserdata = "Userdata"
var UserMessage = ""

// Function to get all data from User by Session
//  Params: Session(String) -> Get data from specific user
//  Return: UserData(type from model) -> Struct with data of the user
func GetUserBySession(session string) UserData {
                result := UserData{}
                if session != "" {
                                database := db.OpenConnection()
                                coll := database.C(DbCollUserdata)
                                coll.Find(bson.M{"Session": session}).One(&result)
                                db.CloseConnection()
                }
                return result
}

// Function to get all data from User by username
//  Params: Username(String) -> Get data from specific user
//  Return: UserData(type from model) -> Struct with data of the user
func GetUserByUname(uname string) UserData {
                result := UserData{}
                if uname != "" {
                                database := db.OpenConnection()
                                coll := database.C(DbCollUserdata)
                                coll.Find(bson.M{"Username": uname}).One(&result)
                                db.CloseConnection()
                }
                return result
}

// Function to a title(like admin/guest) to a user
//  Params: Request, Title(String) -> Set title to user by session
func SetTitle(r *http.Request, title string){
                user := GetUserData(r)
                database := db.OpenConnection()
                coll := database.C(DbCollUserdata)
                coll.Update(user, bson.M{"$set": bson.M{ "Title" : title }})
                db.CloseConnection()
}

// Function to a avatar image to a user
//  Params: Request, Path(String) -> Set avatar to user by session into db
func SetAvatar(r *http.Request, path string){
                user := GetUserData(r)
                database := db.OpenConnection()
                coll := database.C(DbCollUserdata)
                coll.Update(user, bson.M{"$set": bson.M{ "Avatar" : path }})
                db.CloseConnection()
}

// Function to delete the user
// Deleting Avatar, Events/Relations, Session, User
//  Params: ResponseWriter, Request -> For execute
func DeleteUser(w http.ResponseWriter, r *http.Request){
		database := db.OpenConnection()
		coll := database.C(DbCollUserdata)
		user := GetUserData(r)
		coll.Remove(bson.M{ "Session" : user.SessionId })
		// Logout
		db.CloseConnection()
		login.DeleteCookie(w)
		// Delete avatar and events/relations
		DeleteAvatar(user.Avatar)
		DelEventsById(user.Id)
}

// Function to get all data from current User (Session)
//  Params: Request -> Get data from user by session
//  Return: UserData(type from model) -> Struct with data of the user
func GetUserData(r *http.Request) UserData{
		cookie, _ := r.Cookie(CookieName)
		data := UserData{}
		if cookie != nil {
				data = GetUserBySession(cookie.Value)
		}
		return data
}

// Function to get all data from User
//  Return: []UserData(type from model) -> Struct arrays with data of the users
func GetAllUserData() []UserData{
		data := []UserData{}
		database := db.OpenConnection()
		coll := database.C(DbCollUserdata)
		coll.Find(nil).All(&data)
		db.CloseConnection()
		return data
}
```

STRUCT.MODEL.GO

```
// Class for struct handling
// This class just provide public struct for models and work with private helper structs
// With this help, other classes do not need imports for use of structs
package models

import (
		"gopkg.in/mgo.v2/bson"
)

// An UserData represents a user with additional data
type UserData struct{
		Id			bson.ObjectId		`bson:"_id"`
		Username string		`bson:"Username"`
		Title string		`bson:"Title"`
		Password string		`bson:"Password"`
		SessionId string		`bson:"Session"`
		Avatar string				`bson:"Avatar"`
}

// An Device represents a device with additional data
type Device struct{
		Id				bson.ObjectId		`bson:"_id"`
		Name		string				`bson:"Name"`
		Room		string				`bson:"Room"`
		Type		string				`bson:"Type"`
		State		int						`bson:"State"`
}

// An Event represents a event with additional data
type Event struct{
		Id				bson.ObjectId		`bson:"_id"`
		UserId		bson.ObjectId		`bson:"UserId"`
		Name		string				`bson:"Name"`
		Time		string				`bson:"Time"`
		Offset  string				`bson:"Offset"`
}

// An Type represents a type with additional data
type Type struct{
		Id			bson.ObjectId		`bson:"_id"`
		Name		string				`bson:"Name"`
		Kind		string				`bson:"Kind"`
		Min			int							`bson:"Min"`
		Max			int							`bson:"Max"`
}

// An RelationEventDevice represents a relation between Event and Devices and the new state
// With this type joins will be build
type RelationEventDevice struct{
		Id				bson.ObjectId		`bson:"_id"`
		EventId		bson.ObjectId		`bson:"EventId"`
		DeviceId	bson.ObjectId		`bson:"DeviceId"`
		NewState	int				`bson:"NewState"`
}

// An AllDTE is a type arrays of all items
type AllDTE struct{
		Devices []Device
		Events		[]Event
		Types	[]Type
		Rel	[]Item
}

// An Item is a type with data from devices and event actions
type Item struct{
		Id			bson.ObjectId
		Name		string
		Room		string
		Value		int
}

// An Item is a type with arrays of device names and event actions
type Items struct{
		Name		[]string
		Value		[]string
}

type SimState struct {
		Id	bson.ObjectId `bson:"_id" json:"Id" xml:"id"`
		CurrentTime int64		`json:"Time" xml:"time" bson:"time"`
		Sunrise	string	`json:"Sunrise" xml:"sunrise"`
		Sunset	string	`json:"Sunset" xml:"sunset"`
		State		bool	`json:"State" xml:"state"`
		Multiplier  int	`json:"Multiplier" xml:"multiplier"`
}

// Contains information to export the whole database
type xmlData struct {
		Devices		[]Device		`xml:"devices"`
		Types			[]Type			`xml:"types"`
		Events		[]Event			`xml:"events"`
		Relations	[]RelationEventDevice`xml:"relations"`
		Simulator	[]SimState		`xml:"simulator"`
		Users		[]UserData	`xml:"users"`
}
```

SIMULATOR.MODEL.GO

```
package models

import (
		"os"
		"io"
		"fmt"
		"net/http"
		"encoding/xml"
		"strings"
		"io/ioutil"
		"toHuus/db"
)

// Initialisation
```

```
const tmpPath = "./toHuus/conf/tmp/"
const importButton = "dataFile"

// Function to import an XML to DB
//  Params: ResponseWriter, Request -> Image by ParseMultipartForm
func Import(w http.ResponseWriter, r *http.Request) {
	// Get
	r.ParseMultipartForm(32 << 20)
	file, handler, err := r.FormFile(importButton)
	if err != nil {
		fmt.Println(err)
		return
	}
	f, err := os.Create(tmpPath + handler.Filename)
	// Close
	if err != nil {
		fmt.Println(err)
		return
	}
	// Write and read (rename)
	io.Copy(f, file)
	f.Close()
	file.Close()
	newPath := tmpPath + "toHuus.xml"
	os.Rename(tmpPath + handler.Filename, newPath)
	bytes, _ := ioutil.ReadFile(newPath)
	// Parse
	data := xmlData{}
	xml.Unmarshal(bytes, &data)
	// DB
	database := db.OpenConnection()
	// Insert
	for _, e := range data.Devices {
		database.C(dbCollDevices).Insert(e)
	}
	database.C(dbCollTypes).DropCollection() // Reset types
	for _, e := range data.Types {
		database.C(dbCollTypes).Insert(e)
	}
	for _, e := range data.Relations {
		database.C(dbCollRelEvents).Insert(e)
	}
	for _, e := range data.Events {
		database.C(dbCollEvents).Insert(e)
	}
	database.C(dbCollSim).DropCollection() // Reset simulator
	for _, e := range data.Simulator {
		database.C(dbCollSim).Insert(e)
	}
	for _, e := range data.Users {
		database.C(DbCollUserdata).Insert(e)
	}
	// Maybe want to remove file
	//os.Remove(newPath)
}

// Function to export an XML from DB
//  Params: ResponseWriter, Request -> Return xml for download
func Export(w http.ResponseWriter, r *http.Request) {
	// Create
	data := xmlData{}
	data.Devices = GetAllDevices()
	data.Types = GetAllTypes()
	data.Events = GetAllEvents("")
	data.Relations = GetAllRelation()
	data.Simulator = GetSimData()
	data.Users = GetAllUserData()
	// Define
	w.Header().Set("Content-Disposition", "attachment; filename=toHuus.xml")
	w.Header().Set("Content-Type", "application/octet-stream")
	w.Header().Add("Access-Control-Expose-Headers", "Content-Disposition")
	// Parse and send
	xml, _ := xml.Marshal(data)
	io.Copy(w, strings.NewReader(string(xml)))
}
```

INTERFACE.MODEL.GO

```
// Class for interface/basic manipulation
// This class processing basic data or calling functions
// Actions from controller check more specific by data and put to correct function
package models

import (
		"os"
		"net/http"
		"io"
		"strings"
		"fmt"
)

// Initialisation
const avatarPath = "./toHuus/conf/avatar/"
const avatarPathHTML = "./avatar/"
const avatarNameAdditive = "avatar_"
const avatarButton = "avatarFile"

// Function to add an new data by calling specific functions
//  Params: Request, dataType(String) -> Item data from form
func AddData(r *http.Request, dataType string){
		switch dataType{
		case "devices":
				// Add a device
				AddDevice(r.FormValue("dName"), r.FormValue("dRoom"), r.FormValue("dType"))
				break
		case "types":
				// Add a type
				AddType(r.FormValue("tName"), r.FormValue("tKind"), r.FormValue("tMin"), r.FormValue("tMax"))
				break
		case "events":
				// Add a event
				deviceItems := getDeviceFormForEvent(r)
				AddEvent(r.FormValue("eName"), r.FormValue("eTime"), r.FormValue("eOffset"), deviceItems, GetUserData(r).Id)
				break
		default:
				UserMessage = "Error: Can not add data"
		}
}

// Function to update data by calling specific functions
//  Params: Request, dataType(String) -> Item data from form
func UpdateData(r *http.Request, dataType string){
		switch dataType{
		case "devices":
				// Update a device
				UpdateDevice(r.FormValue("dName"), r.FormValue("dRoom"), r.FormValue("dType"))
				break
		case "types":
				// Update a type
				UpdateType(r.FormValue("tName"), r.FormValue("tKind"), r.FormValue("tMin"), r.FormValue("tMax"))
				break
		case "events":
				// Update an event
				deviceItems := getDeviceFormForEvent(r)
				UpdateEvent(r.FormValue("eName"), r.FormValue("eTime"), r.FormValue("eOffset"), deviceItems, GetUserData(r).Id)
				break
		default:
				UserMessage = "Error: Can not update data"
		}
}

// Helper function to get all devices with values from a form
//  Params: Request -> Process From data
func getDeviceFormForEvent(r *http.Request) Items{
		deviceItems := Items{}
		// Get all devices out of form for the event
		for k, v := range r.Form {
				if k == "eDevice" {
						deviceItems.Name = v
				}
				if k == "to" {
```

```
				deviceItems.Value = v
			}
	}
	return deviceItems
}

// Function to delete data by calling specific functions
//  Params: Request, Name(Strign), dataType(String) -> Item data from form and by name
func DelData(r *http.Request, name string, dataType string){
	switch dataType{
	case "devices":
		// Delete an device
		DelDevice(name)
		break
	case "types":
		// Delete an type
		DelType(name)
		break
	case "events":
		// Delete an event
		DelEvent(name, GetUserData(r).Id)
		break
	default:
		UserMessage = "Error: Can not delete data"
	}
}

// Function to upload an avatar image and write to config files
//  Params: ResponseWriter, Request -> Image by ParseMultipartForm
func UploadAvatar(w http.ResponseWriter, r *http.Request) {
	// Get
	r.ParseMultipartForm(32 << 20)
	file, handler, err := r.FormFile(avatarButton)
	if err != nil {
		fmt.Println(err)
		return
	}
	defer file.Close()
	user := GetUserData(r)
	f, err := os.Create(avatarPath + handler.Filename)
	// Close
	if err != nil {
		fmt.Println(err)
		return
	}
	io.Copy(f, file)
	f.Close()
	// Rename
	ending := strings.SplitAfter(handler.Filename, ".")
	newPath := avatarPath + avatarNameAdditive + user.Username + "." + ending[len(ending)-1]
	newPathHTML := avatarPathHTML + avatarNameAdditive + user.Username + "." + ending[len(ending)-1]
	os.Rename(avatarPath + handler.Filename, newPath)
	// Update User
	SetAvatar(r, newPathHTML)
}

// Function to delete an avatar image
//  Params: Path(String) -> Path to file
func DeleteAvatar(path string) {
	f, err := os.Create(path)
	f.Close()
	// Close
	if err != nil {
		fmt.Println(err)
		return
	}else{
		err = os.Remove(path)
		if err != nil{
			UserMessage = "Error: " + err.Error()
			return
		}
	}
}
```

```
// Class for data manipulation
// This class processing data of items (Add/Edit/Delete/Get)
// Package db is needed
package models

import (
		"gopkg.in/mgo.v2/bson"
		"toHuus/db"
		"strconv"
		"strings"
)

// Initialisation
const dbCollDevices = "Devices"
const dbCollEvents = "Events"
const dbCollTypes = "Types"
const dbCollRelEvents = "RelationEventsDevices"
const dbCollSim = "Simulator"

// Function to add an new device to db
//  Params: Name(String), Room(String), dataType(String) -> Device data
func AddDevice(name string, room string, dataType string){
		database := db.OpenConnection()
		coll := database.C(dbCollDevices)
		device := Device{}
		coll.Find(bson.M{"Name" : name}).One(&device)
		// Check not present
		if len(device.Name) < 1 && len(name) > 0 && len(room) > 0 {
				// Write to db
				device = Device{
						bson.NewObjectId(),
						name,
						room,
						dataType,
						0,
				}
				coll.Insert(device)
		} else {
				UserMessage = "Error: Device already exist"
		}
		db.CloseConnection()
}

// Function to add an new event to db for an specific user
//  Params: Name(String), Time(String), dataType(Items), userid(ObjectId) -> Event data
func AddEvent(name string, time string, offset string, deviceItems Items, userid bson.ObjectId){
		database := db.OpenConnection()
		coll := database.C(dbCollEvents)
		coll2 := database.C(dbCollRelEvents)
		event := Event{}
		eventid := bson.NewObjectId()
		coll.Find(bson.M{"Name" : name}).One(&event)
		// Check not present
		if len(event.Name) < 1 && len(name) > 0 {
				// Write Devices/Relation
				for n := range deviceItems.Name{
						state, _ := strconv.Atoi(deviceItems.Value[n])
						name := strings.Split(deviceItems.Name[n], " | ")[0]
						rel := RelationEventDevice{
								bson.NewObjectId(),
								eventid,
								GetDeviceByName(name).Id,
								state,
						}
						coll2.Insert(rel)
				}
				// Write to db
				event = Event{
						eventid,
						userid,
						name,
						time,
						offset,
				}
				coll.Insert(event)
		} else {
				UserMessage = "Error: Event already exist"
		}
		db.CloseConnection()
}

// Function to add an new tye to db
//  Params: Name(String), Kind(String), Min(String), Max(String) -> Type data
func AddType(name string, kind string, min string, max string){
		database := db.OpenConnection()
		coll := database.C(dbCollTypes)
		// Convert to int
		iMin,err := strconv.Atoi(min)
		iMax,err2 := strconv.Atoi(max)
		if err2 != nil && err != nil {
				// Set switch values
				iMin = 0
				iMax = 1
		}
		// Check not present
		ntype := Type{}
		coll.Find(bson.M{"Name" : name}).One(&ntype)
		if len(ntype.Name) < 1 && len(name) > 0 && iMin < iMax {
				// Write to db
				ntype = Type{
						bson.NewObjectId(),
						name,
						kind,
						iMin,
						iMax,
				}
				coll.Insert(ntype)
		} else {
				UserMessage = "Error: Type already exist"
		}
		db.CloseConnection()
}

// Function to update an device to db
//  Params: Name(String), Room(String), dataType(String) -> Device data
func UpdateDevice(name string, room string, dataType string){
		database := db.OpenConnection()
		coll := database.C(dbCollDevices)
		device := Device{}
		coll.Find(bson.M{"Name" : name}).One(&device)
		// Write to db
		device2 := Device{
				device.Id,
				device.Name,
				room,
				dataType,
				device.State,
		}
		coll.Update(device, device2)
		db.CloseConnection()
}

// Function to update an device to db
//  Params: Name(String), Room(String), dataType(String) -> Device data
func UpdateState(name string, state string){
		database := db.OpenConnection()
		coll := database.C(dbCollDevices)
		device := Device{}
		coll.Find(bson.M{"Name" : name}).One(&device)
		newState, err := strconv.Atoi(state)
		if err == nil {
				// Write to db
				coll.Update(bson.M{"_id" : device.Id}, bson.M{"$set" : bson.M{"State" : newState}})
		}
		db.CloseConnection()
}

// Function to update an event to db for an specific user
//  Params: Name(String), Time(String), dataType(String), userid(ObjectId) -> Event data
func UpdateEvent(name string, time string, offset string, deviceItems Items, userid bson.ObjectId){
		database := db.OpenConnection()
		coll := database.C(dbCollEvents)
```

```
	coll2 := database.C(dbCollRelEvents)
	event := Event{}
	coll.Find(bson.M{"Name" : name}).One(&event)
	// Devices/Relation
	for n := range deviceItems.Name{
		state, _ := strconv.Atoi(deviceItems.Value[n])
		name := strings.Split(deviceItems.Name[n], " | ")[0]
		rel := RelationEventDevice{}
		coll2.Find(bson.M{"EventId" : event.Id, "DeviceId" : GetDeviceByName(name).Id}).One(&rel)
		// Update relation from edit
		if rel.Id != "" {
			rel2 := RelationEventDevice{
				rel.Id,
				event.Id,
				GetDeviceByName(name).Id,
				state,
			}
			coll2.Update(rel, rel2)
		}else{
			// Add new relation from edit
			rel2 := RelationEventDevice{
				bson.NewObjectId(),
				event.Id,
				GetDeviceByName(name).Id,
				state,
			}
			coll2.Insert(rel2)
		}
	}
	// Write to db
	event2 := Event{
		event.Id,
		userid,
		name,
		time,
		offset,
	}
	coll.Update(event, event2)
	db.CloseConnection()
}

// Function to update an tye to db
//  Params: Name(String), Kind(String), Min(String), Max(String) -> Type data
func UpdateType(name string, kind string, min string, max string){
	database := db.OpenConnection()
	coll := database.C(dbCollTypes)
	// Check not present
	ntype := Type{}
	coll.Find(bson.M{"Name" : name}).One(&ntype)
	iMin,err := strconv.Atoi(min)
	iMax,err2 := strconv.Atoi(max)
	if err2 != nil && err != nil {
		// Set switch values
		iMin = ntype.Min
		iMax = ntype.Max
	}
	// Write to db
	ntype2 := Type{
		ntype.Id,
		name,
		kind,
		iMin,
		iMax,
	}
	coll.Update(ntype, ntype2)
	db.CloseConnection()
}

// Function to delete an device from db
//  Params: Name(String) -> Device name to find item
func DelDevice(name string){
	database := db.OpenConnection()
	coll := database.C(dbCollDevices)
	coll.Remove(bson.M{"Name" : name})
	db.CloseConnection()
}
```

```
// Function to delete an event from db
//  Params: Name(String), userId(ObjectId) -> Event name to find item of an specific user
func DelEvent(name string, userid bson.ObjectId){
	database := db.OpenConnection()
	// Device/Relation
	coll := database.C(dbCollRelEvents)
	coll.RemoveAll(bson.M{"EventId" : GetEventByName(name, userid).Id})
	// Events
	coll2 := database.C(dbCollEvents)
	coll2.Remove(bson.M{"Name" : name})
	db.CloseConnection()
}

// Function to delete an type from db
//  Params: Name(String) -> Type name to find item
func DelType(name string){
	database := db.OpenConnection()
	coll := database.C(dbCollTypes)
	coll.Remove(bson.M{"Name" : name})
	db.CloseConnection()
}

// Function to delete multiple events from db
//  Params: userId(ObjectId) -> Delete all events by user
func DelEventsById(userid bson.ObjectId){
	database := db.OpenConnection()
	coll := database.C(dbCollRelEvents)
	coll.RemoveAll(bson.M{"UserId" : userid})
	db.CloseConnection()
}

// Function to get every items from all data
//  Params: userId(ObjectId) -> Delete all events by user
//  Return: AllDTE(type from model) -> Struct with all items
func GetAllDTE(userid bson.ObjectId) AllDTE{
	result := AllDTE{
		GetAllDevices(),
		GetAllEvents(userid),
		GetAllTypes(),
		GetRelationByUser(userid),
	}
	return result
}

// Function to get all devices
//  Return: Devices(type from model) -> Struct with arrays of Device(type)
func GetAllDevices() []Device{
	result := []Device{}
	database := db.OpenConnection()
	coll := database.C(dbCollDevices)
	coll.Find(nil).All(&result)
	db.CloseConnection()
	return result
}

// Function to get all events by user
//  Params: userId(ObjectId) -> Events limited by active user
//  Return: Events(type from model) -> Struct with arrays of Event(type)
func GetAllEvents(userid bson.ObjectId) []Event{
	result := []Event{}
	database := db.OpenConnection()
	coll := database.C(dbCollEvents)
	if userid == ""{
		coll.Find(nil).All(&result)
	}else{
		coll.Find(bson.M{"UserId" : userid}).All(&result)
	}
	db.CloseConnection()
	return result
}

// Function to get all types
//  Return: Types(type from model) -> Struct with arrays of Type(type)
func GetAllTypes() []Type{
	result := []Type{}
```

```
		database := db.OpenConnection()
		coll := database.C(dbCollTypes)
		coll.Find(nil).All(&result)
		db.CloseConnection()
		return result
}

// Function to get all devices by type
//  Params: Type(String) -> Kind of type to find
//  Return: Devices(type from model) -> Struct with arrays of Device(type)
func GetAllDevicesByType(ntype string) []Device{
		result := []Device{}
		database := db.OpenConnection()
		coll := database.C(dbCollDevices)
		coll.Find(bson.M{"Type" : ntype}).All(&result)
		db.CloseConnection()
		return result
}

// Function to get all relations
//  Return: RelationEventDevice(type from model) -> Struct all relation data
func GetAllRelation() []RelationEventDevice{
		relation := []RelationEventDevice{}
		database := db.OpenConnection()
		coll := database.C(dbCollRelEvents)
		coll.Find(nil).All(&relation)
		db.CloseConnection()
		return relation
}

// Function to get all relations to an event
//  Params: EventId(String) -> Event to query
//  Return: Devices(type from model) -> Struct with arrays of Device(type)
func GetRelationToEvent(id string) []Device{
		relation := []RelationEventDevice{}
		resultD := []Device{}
		database := db.OpenConnection()
		coll := database.C(dbCollRelEvents)
		coll.Find(bson.M{"EventId" : id}).All(&relation)
		coll = database.C(dbCollDevices)
		// Check all devices to the id took from relation
		for i := 0; i < len(relation); i++ {
				coll.Find(bson.M{"_id" : relation[i].DeviceId}).One(&resultD[i])
		}
		db.CloseConnection()
		return resultD
}

// Function to get new state for event
//  Params: EventId(String), DeviceId(String) -> Event to query
//  Return: int -> State
func GetNewState(evt string, dev string) int{
		relation := RelationEventDevice{}
		database := db.OpenConnection()
		coll := database.C(dbCollRelEvents)
		coll.Find(bson.M{"EventId" : evt , "DeviceId" : dev}).One(&relation)
		return relation.NewState
}

// Function to get all relations to an user
//  Params: UserId(ObjectId) -> User to query
//  Return: Item(type from model) -> Struct with relation and device data
func GetRelationByUser(id bson.ObjectId) []Item{
		relation := []Item{}
		events := GetAllEvents(id)
		database := db.OpenConnection()
		coll := database.C(dbCollRelEvents)
		// Loop through all events from the user
		for i := 0; i < len(events); i++ {
				buffer := []RelationEventDevice{}
				coll.Find(bson.M{"EventId" : events[i].Id}).All(&buffer)
				// Loop through all relations to get the correct relation in an array
				for j := 0; j < len(buffer); j++ {
						dev := GetDeviceById(buffer[j].DeviceId)
						relation = append(relation,
								Item{ events[i].Id, dev.Name, dev.Room, buffer[j].NewState })
```

```
			}
		}
		db.CloseConnection()
		return relation
}

// Function to get a device by name
//  Params: Name(String) -> Device to query
//  Return: Device(type from model) -> Struct witch device data
func GetDeviceByName(name string) Device{
		result := Device{}
		database := db.OpenConnection()
		coll := database.C(dbCollDevices)
		coll.Find(bson.M{"Name" : name}).One(&result)
		db.CloseConnection()
		return result
}

// Function to get a event by name
//  Params: Name(String), UserId(ObjectId) -> Event to query for a user
//  Return: Event(type from model) -> Struct witch event data
func GetEventByName(name string, userid bson.ObjectId) Event{
		result := Event{}
		database := db.OpenConnection()
		coll := database.C(dbCollEvents)
		coll.Find(bson.M{"Name" : name, "UserId" : userid}).One(&result)
		db.CloseConnection()
		return result
}

// Function to get a device by id
//  Params: Id(String) -> Device to query
//  Return: Device(type from model) -> Struct witch device data
func GetDeviceById(id bson.ObjectId) Device{
		result := Device{}
		database := db.OpenConnection()
		coll := database.C(dbCollDevices)
		coll.Find(bson.M{"_id" : id}).One(&result)
		db.CloseConnection()
		return result
}

// Function to get the kind of an type
//  Params: ntype(String) -> Type name
//  Return: String -> Kind
func GetKindByType(ntype string) string{
		result := Type{}
		database := db.OpenConnection()
		coll := database.C(dbCollTypes)
		coll.Find(bson.M{"Name" : ntype}).One(&result)
		db.CloseConnection()
		return result.Kind
}

// Function to get a event by id
//  Params: Id(String) -> Event to query
//  Return: Event(type from model) -> Struct witch event data
func GetEventById(id string, userid bson.ObjectId) Event{
		result := Event{}
		database := db.OpenConnection()
		coll := database.C(dbCollEvents)
		coll.Find(bson.M{"_id" : id, "UserId" : userid}).One(&result)
		db.CloseConnection()
		return result
}

// Function to get the current sim states as array
//  Return: SimState(type from model) -> Struct witch simulator data
func GetSimData() []SimState{
		result := []SimState{}
		database := db.OpenConnection()
		coll := database.C(dbCollSim)
		coll.Find(nil).All(&result)
		db.CloseConnection()
		return result
}
```

```
// Function to get the current sim states
//  Return: SimState(type from model) -> Struct witch simulator data
func SetSimData(states SimState) {
	result := []SimState{}
	database := db.OpenConnection()
	coll := database.C(dbCollSim)
	coll.Find(nil).All(&result)
	if len(result)>0{
		if result[0].Id == "" {
			coll.Insert(states)
		}else{
			states.Id = result[0].Id
			coll.Update(result[0], states)
		}
	}else{
		coll.Insert(states)
	}
	db.CloseConnection()
}
```

LOGIN.GO

```
// Package for a login
// Call the CheckLogin function
// The construction of HTML forms and inputs is important for this (variables)
// Todo: Enable security again (hash)
package login

import (
	"net/http"
	"fmt"
	"sync"
	"strconv"
	"encoding/hex"
	"crypto/rand"
	"encoding/base64"
	"gopkg.in/mgo.v2"
	"gopkg.in/mgo.v2/bson"
	"toHuus/db"
)

// Initialisation
const dbCollName = "Userdata"
const cookieName = "session"
const buttonName = "authBtn"
const usernameName = "uname"
const passwordName = "passwd"
var formButtons = []string  { "Login", "Registration", "Logout" }
var storageMutex sync.RWMutex
var Message string  // Message returned by calling functions

// An User represents a user with basic data
type User struct {
	Id		bson.ObjectId		`bson:"_id"`
	Username string		`bson:"Username"`
	Title string		`bson:"Title"`
	Password string		`bson:"Password"`
	SessionId string		`bson:"Session"`
	Avatar string			`bson:"Avatar"`
}

// Basic function for this package
//  Params: ResponseWriter, Request -> For cookie handling
//  Return: Boolean -> Action was valid
func CheckLogin(w http.ResponseWriter, r *http.Request) bool{
	var valid bool
	database := db.GetConnection()
	coll := database.C(dbCollName)
	button := r.PostFormValue(buttonName) // r.Form[]
	if len(button) <= 2 {
		// Check for cookies to handle session, because no action found
		valid = CheckCookie(w, r)
	}else{
		uname := r.PostFormValue(usernameName)
		password := r.PostFormValue(passwordName)
```

```
				switch button {
				case formButtons[0]:
					// Call login
					valid = login(w, r, uname, password, coll)
				case formButtons[1]:
					// Call registration
					valid = register(uname, password, coll)
				case formButtons[2]:
					// Call logout
					logout(w, uname, coll)
					valid = false
				default:
					valid = false
					Message = "Error: Unknown"
				}
		}
		return valid
}

// Basic function for sessions
//  Params: ResponseWriter, Request -> For cookie handling
//  Return: Boolean -> Cookie is valid
func CheckCookie(w http.ResponseWriter, r *http.Request) bool{
		var valid bool
		cookie, err := r.Cookie(cookieName)
		database := db.GetConnection()
		coll := database.C(dbCollName)
		if err != nil {
				if err != http.ErrNoCookie {
					fmt.Fprint(w, err)
					valid = false
					Message = "Error: No Session"
				} else {
					err = nil
				}
		}
		if cookie != nil {
				// Check session is valid
				result := User{}
				storageMutex.RLock()
				coll.Find(bson.M{ "Session" : cookie.Value }).One(&result)
				storageMutex.RUnlock()
				if result.SessionId != "" {
					valid = true
				}
		}else{
				valid = false
		}
		return valid
}

// Function to handle registration
//  Params: Username(String), Password(String), Collection -> Get username and password
//  Return: Boolean -> Action was valid
func register(uname string, password string, coll *mgo.Collection) bool{
		var valid bool
		// Check if the input of username and password is valid
		if validatePassword(password) && validateUsername(uname) {
				// Insert to database
				coll.Insert(User{
					bson.NewObjectId(),
					uname,
					"",
					password, // hash(password) | disabled security
					"",
					"",
				})
				Message = "Rigistered"
				valid = false
		}else{
				valid = false
				if Message == "" {
					Message = "Successfully registered"
				}
		}
		return valid
```

```
}

// Function to handle login
//  Params: ResponseWriter, Request, Username(String), Password(String), Collection -> Get cookie, username and password
//  Return: Boolean -> Action was valid
func login(w http.ResponseWriter, r *http.Request, uname string, password string, coll *mgo.Collection) bool{
	var valid bool
	result := User{}
	coll.Find(bson.M{ "Username" : uname }).One(&result)
	// Check if username and password from input is valid with db
	if result.Username == uname && result.Password == password { // hash(password) | disabled security
		// Create session
		sessionId := generateSessionId()
		coll.Update(result, bson.M{"$set": bson.M{ "Session" : sessionId }})
		DeleteCookie(w)
		setCookie(w, r, sessionId, coll)
		valid = true
		Message = "Successfully logged in"
	}else{
		valid = false
		if len(Message) <= 2 {
			Message = "Error: Invalid username or password"
		}
	}
	return valid
}

// Function to handle logout
//  Params: ResponseWriter, Username(String), Collection -> Get cookie and username
func logout(w http.ResponseWriter, uname string, coll *mgo.Collection){
	// Delete the session and cookie
	result := User{}
	coll.Find(bson.M{ "Username" : uname }).One(&result)
	coll.Update(result, bson.M{ "$set": bson.M{ "Session" : "" }})
	DeleteCookie(w)
	Message = "Logged out"
}

// Helper-Function to delete an cookie of w
//  Params: ResponseWriter -> Get cookie
func DeleteCookie(w http.ResponseWriter)  {
	newCookie := http.Cookie{
		Name: cookieName,
		MaxAge: -1,
	}
	http.SetCookie(w, &newCookie)
}

// Function to set a new cookie to user
//  ResponseWriter, Request, session id(String), Collection -> Get cookie and session
func setCookie(w http.ResponseWriter, r *http.Request, sessionId string, coll *mgo.Collection) {
	// Check for present cookie
	cookie, err := r.Cookie(cookieName)
	if err != nil {
		if err != http.ErrNoCookie {
			fmt.Fprint(w, err)
			return
		} else {
			err = nil
		}
	}
	// Generate a new session
	if sessionId == "" {
		sessionId = generateSessionId()
	}
	// Set cookie to user and db
	cookie = &http.Cookie{
		Name: cookieName,
		Value: sessionId,
	}
	result := User{}
	storageMutex.Lock()
	coll.Find(bson.M{ "Session" : cookie.Value }).One(&result)
	coll.Update(result, bson.M{"$set": bson.M{ "Session" : sessionId }})
	storageMutex.Unlock()
	http.SetCookie(w, cookie)
```

```
}

// Helper-Function to generate an session
//  Return: String -> Session id
func generateSessionId() string{
		buffer := make([]byte, 32)
		// Random byte
		_, err := rand.Read(buffer)
		if err != nil {
				panic(err)
		}
		// Encode byte
		return hex.EncodeToString(buffer)
}

// Helper-Function to check password by rules
//  Param: String -> Password
//  Return: Boolean -> valid
func validatePassword(password string) bool{
		pLenght := 6
		var valid bool
		// Length
		if len(password) >= pLenght {
				valid = true
		}else{
				valid = false
				Message = "Error: Invalid password length (min. " + strconv.Itoa(pLenght) + ")"
		}
		return valid
}

// Helper-Function to check username by rules
//  Param: String -> Username
//  Return: Boolean -> valid
func validateUsername(uname string) bool{
		uLenght := 4
		var valid bool
		// Length
		if len(uname) >= uLenght {
				// Check username forgiven
				database := db.GetConnection()
				coll := database.C(dbCollName)
				result := User{}
				coll.Find(bson.M{ "Username" : uname }).One(&result)
				if result.Username != uname {
						valid = true
				}else{
						valid = false
						Message = "Error: User already exist"
				}
		}else{
				valid = false
				Message = "Error: Invalid username length (min. " + strconv.Itoa(uLenght) + ")"
		}
		return valid
}

// Helper-Function to hash an password for security
//  Param: String -> Clean password
//  Return: String -> Hashed string
func hash(data string) string{
		return base64.StdEncoding.EncodeToString([]byte(data))
}
```

DB.GO

```
// Package for a database connection
// Call the Database function at start once
// Call OpenConnection to get the database pointer
package db

import (
		"gopkg.in/mgo.v2"
		"fmt"
)
```

```
// Initialisation
var url = ""
var name = ""
var db *mgo.Database = nil

// Basic function for this package to initialise the connection
// Call once at program start
//  Params: String, String -> DB Name and DB URL
func Database(dbName string, dbUrl string){
	setDB(dbName)
	setUrl(dbUrl)
}

// Function to open an connection from the db
//  Return: *Database -> Pointer of DB connection
func OpenConnection() *mgo.Database{
	// Open
	session, err := mgo.Dial(url)
	if err != nil {
		fmt.Println(err)
	}
	db = session.DB(name)
	//defer
	return db
}

// Function to close an connection from the db
func CloseConnection(){
	db.Session.Close()
}

// Function to get the connection
//  Return: *Database -> Pointer of DB connection
func GetConnection() *mgo.Database{
	return db
}

// Helper-Function to set name
//  Params: String -> DB Name
func setDB(dbName string){
	name = dbName
}

// Helper-Function to set url
//  Params: String -> DB  url
func setUrl(dbUrl string){
	url = dbUrl
}
```

STRUCT.CONTROLLER.GO

```
// Class for struct handling
// This class just provide public struct for controllers and work with private helper structs
// With this help, other classes do not need imports for use of structs
// Package models is needed
package controllers

import (
	"toHuus/models"
)

// A Load represents a type needed to build templates with additional data like Nav(Struct)
type Load struct{
	Nav			Nav
	Message	string
	User			models.UserData
}

// A Nav represents an array with Nav elements
type Nav struct{
	Elements	[]NavElement
}

// A NavElement represents a type with data for the nav
type NavElement struct{
	Name		string
```

```
		Ref		string
		Icon		string
}
```

SIMULATOR.CONTROLLER.GO

```
// Class for handling simulator actions
// This class check und serialize actions for the simulator
// Package db, login, models, simulator is needed
package controllers

import (
		"net/http"
		"text/template"
		"encoding/json"
		"time"
		"strconv"
		"gopkg.in/mgo.v2/bson"
		"toHuus/login"
		"toHuus/models"
		"toHuus/db"
		"strings"
)

// Function to show the simulator interface
//  Params: ResponseWriter, Request -> For execute
func ShowSimulator(w http.ResponseWriter, r *http.Request){
		// Load templates
		t := template.Must(template.ParseFiles("./toHuus/views/header.html",
				"./toHuus/views/simulator.html", "./toHuus/views/footer.html"))
		// Executes templates with data (Nav, Message, Userdata)
		t.ExecuteTemplate(w, "header",
				Load{getSimulatorNav(), models.UserMessage, models.GetUserData(r)})
		t.ExecuteTemplate(w, "content", nil)
		t.ExecuteTemplate(w, "footer", nil)
}

// Basic function to handle the login check and carry out another action to simulator
//  Params: ResponseWriter, Request -> For execute
func SimulatorHandler(w http.ResponseWriter, r *http.Request){
		// DB
		db.OpenConnection()
		// Check Cookies
		if login.CheckCookie(w,r) {
				// Show simulator
				if login.Message != "" {
						models.UserMessage = login.Message
				}
				ShowSimulator(w,r)
		}else{
				// Redirect to the interface/login
				http.Redirect(w, r, url, 301)
		}
		db.CloseConnection()
}

// Function to handle data like import and export for simulator
//  Params: ResponseWriter, Request -> For execute
func DataHandler(w http.ResponseWriter, r *http.Request){
		ntype := r.FormValue("type")
		// Handle Add or Update(Edit)
		if ntype == "import" {
				models.Import(w, r)
				// Return to the sim with an anchor
				http.Redirect(w, r, url + "sim#data", 301)
		} else if ntype == "export" {
				models.Export(w, r)
		}
}

// Function to set data to db for the simulator
//  Params: ResponseWriter, Request -> For execute
func SimSetHandler(w http.ResponseWriter, r *http.Request){
		set := r.FormValue("Set")
		val := r.FormValue("Value")
		data := models.GetSimData()[0]
		if(set == "State"){
			newVal, err := strconv.ParseBool(val)
			if err == nil{
				data.State = newVal
			}
			models.SetSimData(data)
		}else if(set == "Time"){
			// Convert to duration
			hours , _ := strconv.Atoi(strings.Split(val, ":")[0])
			mins , _ := strconv.Atoi(strings.Split(val, ":")[1])
			newVal := (time.Duration(hours)*time.Hour) +
				(time.Duration(mins)*time.Minute)
			data.CurrentTime = newVal.Nanoseconds()
			models.SetSimData(data)
		}else if(set == "Multiplier"){
			newVal, err := strconv.Atoi(val)
			if err == nil{
				data.Multiplier = newVal
			}
			models.SetSimData(data)
		}
}

// Function to handle the get requests of data from simulator got by /ui/sim
// Executes json data
//  Params: ResponseWriter, Request -> For execute
func SimGetHandler(w http.ResponseWriter, r *http.Request){
		var result []byte
		get := r.FormValue("Get")
		// Return devices
		data := models.GetSimData()
		if len(data)>0 {
			if get == "States" {
				result, _ = json.Marshal(data[0])
			}
		}
		w.Header().Set("Content-Type", "application/json")
		w.Write(result)
}

// Helper function to show an specific Nav/Menu in header template
// Specific for every site-handler
//  Return: Nav(type from controller) -> Navigation data (Name, Anchor, Icon)
func getSimulatorNav() Nav{
		elements := []NavElement{
			{"Simulator","simUi", "home"},
			{"Data","data", "cogs"},
			{"About","about", "info"},
		}
		return Nav{elements}
}

// Helper function for main to add default simulator states
func SetSimStates(){
		// Initialise time
		time.LoadLocation("Europe/Berlin")
		// Set default states
		models.SetSimData(models.SimState{bson.NewObjectId(), time.Now().Unix(),
		"", "", false, 1})
}
```

LOGIN.CONTROLLER.GO

```
// Class for handling login actions
// This class check und serialize actions for the login
// Package db, login, models is needed
package controllers

import (
		"html/template"
		"net/http"
		"toHuus/db"
		"toHuus/login"
		"toHuus/models"
)

// Function to show the login
//  Params: ResponseWriter, Request -> For execute
```

```
func ShowLogin(w http.ResponseWriter, r *http.Request){
		// Load templates
		t := template.Must(template.ParseFiles("./toHuus/views/header.html",
			"./toHuus/views/login.html", "./toHuus/views/footer.html"))
		// Executes templates with data (Nav, Message)
		t.ExecuteTemplate(w, "header",
			Load{getLoginNav(), models.UserMessage, models.UserData{}})
		t.ExecuteTemplate(w, "content", nil)
		t.ExecuteTemplate(w, "footer", nil)
}

// Basic function to handle the login check and carry out another action to login
//  Params: ResponseWriter, Request -> For execute
func CheckLogin(w http.ResponseWriter, r *http.Request) {
		// DB
		db.OpenConnection()
		// Check Cookies
		if login.CheckLogin(w,r) {
			// Redirect to UI
			models.UserMessage = login.Message
			http.Redirect(w, r, url + "ui", 301)
		} else {
			// Show Login
			models.UserMessage = login.Message
			ShowLogin(w,r)
		}
		db.CloseConnection()
}

// Helper function to show an specific Nav/Menu in header template
// Specific for every site-handler
//  Return: Nav(type from controller) -> Navigation data (Name, Anchor, Icon)
func getLoginNav() Nav{
		elements := []NavElement{
			{"Login","login", "home"},
			{"Data","data", "cogs"},
			{"About","about", "info"},
		}
		return Nav{elements}
}
```

INTERFACE.CONTROLLER.GO

```
// Class for handling interface actions
// This class check und serialize actions for the interface
// Package db, login, models is needed
package controllers

import (
		"net/http"
		"html/template"
		"encoding/json"
		"toHuus/login"
		"toHuus/models"
		"toHuus/db"
		"os"
		"fmt"
)

// An Config represents the data for database config
type config struct{
		Url				string `json:"url"`
		Db						string `json:"database"`
		Port				string `json:"port"`
}

// Initialisation
var url = UrlConfig()

// Function to show the user interface
//  Params: ResponseWriter, Request -> For execute
func ShowInterface(w http.ResponseWriter, r *http.Request){
		// Load templates
		t := template.Must(template.ParseFiles("./toHuus/views/header.html",
			"./toHuus/views/interface.html", "./toHuus/views/footer.html"))
		// Executes templates with data (Nav, Message, Userdata, AllItemData)
		t.ExecuteTemplate(w, "header",
			Load{getInterfaceNav(), models.UserMessage, models.GetUserData(r)})
		t.ExecuteTemplate(w, "content", models.GetAllDTE(models.GetUserData(r).Id))
		t.ExecuteTemplate(w, "footer", nil)
}

// Basic function to handle the login check and carry out another action to interface
//  Params: ResponseWriter, Request -> For execute
func InterfaceHandler(w http.ResponseWriter, r *http.Request){
		// DB
		db.OpenConnection()
		// Check Cookies
		if(login.CheckCookie(w, r)){
			// Show Interface
			if(login.Message != ""){
				models.UserMessage = login.Message
			}
			ShowInterface(w, r)
		}else{
			// Redirect to Login
			models.UserMessage = login.Message
			http.Redirect(w, r, url, 301)
		}
		db.CloseConnection()
}

// Helper function to show an specific Nav/Menu in header template
// Specific for every site-handler
//  Return: Nav(type from controller) -> Navigation data (Name, Anchor, Icon)
func getInterfaceNav() Nav{
		elements := []NavElement{
			{"Overview","home", "home"},
			{"Devices","devices", "th"},
			{"Events","events", "calendar"},
			{"Types","types", "th-list"},
			{"User","user", "user"},
			{"About","about", "info"},
		}
		return Nav{elements}
}

// Function to handle action at user data got by /ui/user
// After that return to ui
//  Params: ResponseWriter, Request -> For execute
func UserHandler(w http.ResponseWriter, r *http.Request){
		// Get action from button
		set := r.FormValue("set")
		del := r.FormValue("del")
		if set != "" {
			if set == "avatar" {
				models.UploadAvatar(w, r)
			}else if set == "title" {
				models.SetTitle(r, r.FormValue("title"))
			}
		}else if del == "user" {
			models.DeleteUser(w, r)
		}
		http.Redirect(w, r, url + "ui#user", 301)
}

// Function to handle adding/update new items got by /ui/add
// After that return to ui
//  Params: ResponseWriter, Request -> For execute
func AddHandler(w http.ResponseWriter, r *http.Request){
		// Get action from button
		back := "home"
		device := r.FormValue("addDevice")
		group := r.FormValue("addType")
		event := r.FormValue("addEvent")
		// Handle Add or Update(Edit)
		if device == "Add" {
			back = "devices"
			models.AddData(r, back)
		} else if device == "Update" {
			back = "devices"
			models.UpdateData(r, back)
		}
		if group == "Add" {
			back = "types"
			models.AddData(r, back)
		} else if group == "Update" {
			back = "types"
			models.UpdateData(r, back)
		}
		if event == "Add" {
			back = "events"
			models.AddData(r, back)
		} else if event == "Update" {
			back = "events"
			models.UpdateData(r, back)
		}
		// Return to the ui with an anchor
		http.Redirect(w, r, url + "ui#"+back, 301)
}

// Function to handle update state from overview action
//  Params: ResponseWriter, Request -> For execute
func StateHandler(w http.ResponseWriter, r *http.Request) {
		// Get data
		state := r.FormValue("State")
		name := r.FormValue("Name")
		// Update
		models.UpdateState(name, state)
}

// Function to handle the get requests of data from item got by /ui/get
// Executes json data
//  Params: ResponseWriter, Request -> For execute
func GetHandler(w http.ResponseWriter, r *http.Request){
		var result []byte
		get := r.FormValue("Get")
		// Return devices
		if get == "AllDevicesByType" {
			result, _ = json.Marshal(models.GetAllDevicesByType(r.FormValue("Type")))
		}else if get == "AllDevices" {
			// Change type to art for overview
			data := models.GetAllDevices()
			for i := 0; i<len(data); i++ {
				data[i].Type = models.GetKindByType(data[i].Type)
			}
			result, _ = json.Marshal(data)
		}
		w.Header().Set("Content-Type", "application/json")
		w.Write(result)
}

// Function to handle deleting requests got by /ui/del
// After that return to ui
//  Params: ResponseWriter, Request -> For execute
func DelHandler(w http.ResponseWriter, r *http.Request){
		back := "home"
		item := r.FormValue("Item")
		name := r.FormValue("Name")
		if item == "D" {
			back = "devices"
			models.DelData(r, name, back)
		} else if item == "T" {
			back = "types"
			models.DelData(r, name, back)
		} else if item == "E" {
			back = "events"
			models.DelData(r, name, back)
		}
		http.Redirect(w, r, url + "ui#"+back, 301)
}

// Helper function for main to add basic types
func SetDefaultTypes(){
		if len(models.GetAllTypes()) < 1 {
			models.AddType("Light", "Switch", "0", "1")
			models.AddType("Roll Shutter", "Range", "0", "100")
			models.AddType("Dimmer", "Range", "0", "100")
```

```
                models.AddType("Heater", "Number", "0", "6")
                models.AddType("Coffee Maschine", "Switch", "0", "1")
        }
}

// Loading the configuration file for db
func DbConfig() (string, string){
        f, err := os.Open("./toHuus/conf/dbConf.json")
        if err != nil {
                fmt.Println("No file")
        }
        d := json.NewDecoder(f)
        conf := config{}
        err = d.Decode(&conf)
        if err != nil {
                fmt.Println("Bad file")
        }
        return conf.Db, conf.Url + ":" + conf.Port
}

// Loading the configuration file for url to client
func UrlConfig() string{
        f, err := os.Open("./toHuus/conf/urlConf.json")
        if err != nil {
                fmt.Println("No file")
        }
        d := json.NewDecoder(f)
        conf := config{}
        err = d.Decode(&conf)
        if err != nil {
                fmt.Println("Bad file")
        }
        return conf.Url
}
```

DB.CONFIG.JSON

```
{
  "url": "mongodb:// localhost
  "database":  "HA17DB_Sven_Kuhlmann_MN610292"
  "port": "27017"
}
```

URLCONFIG.JSON

```
{
  "url": "http://localhost:4242/"
}
```