NetPaint - Quellcode

WEB-TECHNOLOGIEN

VON FABIAN PUSZIES
MATR.-NR.: 530214

Inhaltsverzeichnis

1. index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
	<meta charset="UTF-8">
	<title>netPaint</title>
	<link rel="stylesheet" href="css/style.css">
	<link rel="stylesheet" href="css/font.css">
</head>
<body>
	<!--
		Hier werden Canvas Elemente eingefügt
	-->
	<div id="canvasWrapper">

	</div>

	<!--
		Das Menu der Applikation
	-->
	<div class="mainOptionsWrapper" id="mainOptionsWrapper">
		<div class="mainOptionsMenu">
			<div class="row">
				<a href="#newCanvas">Neu</a>
				<a href="#" id="storeLink">Bild speichern</a>
				<a href="#wrapperStorage" id="loadImgLink">Bild laden</a>
				<a href="#" id="downloadLink" target="_blank">Download</a>
				<a href="#newGroup">Gruppe erstellen</a>
			</div>
			<div class="row">
				<a href="#" id="saveSettings">Einstellungen speichern</a>
				<a href="#wrapperStorage" id="loadSettings">Einstellungen laden</a>

				<label for="groupSelection">Gruppe auswählen</label>
				<select id="groupSelection">
					<option>Keine</option>
				</select>
			</div>
		</div>
		<div class="openMainOptions">
			<a href="#" class="iconLine" id="openMainOptions"><span class="glyphicon
glyphicon-arrow-down"> Einstellungen</span> </a>
			<a href="#" class="iconLine" id="closeMainOptions"><span class="glyphicon
glyphicon-arrow-up"> Einstellungen</span> </a>
		</div>
	</div>

	<!--
		Das Menu der Stifte
	-->
	<div class="drawOperations" id="drawOperationsMenu">
		<div class="floating openCloseMenu" id="#">
			<a href="#" class="iconLine" id="openDrawOperations"><span class="glyphicon
glyphicon-arrow-left"> Stifte</span> </a>
			<a href="#" class="iconLine" id="closeDrawOperations"><span class="glyphicon
glyphicon-arrow-right"> Stifte</span> </a>
		</div>
		<div class="floating paintOptions">
			<h2>Stifte</h2>
			<p class="penSelectorLabel">Stifte: </p>
			<div id="penSelector">

			</div>
```

```
                        <label for="widthPicker">Stiftbreite: </label>
                        <select class="penOptions" name="strokeWidth" id="widthPicker">
                                <option value="10">10px</option>
                                <option value="15" selected>15px</option>
                                <option value="20">20px</option>
                                <option value="30">30px</option>
                        </select>


                        <a href="#" id="revert">Operation zurück</a>

                        <label for="strokePicker">Zeichenart:</label>
                        <select name="lineJoin" id="strokePicker">
                                <option value="round" selected>rund</option>
                                <option value="bevel">glatt</option>
                                <option value="miter">eckig</option>
                        </select>

                        <label for="colorPicker">Farben wählen:</label>
                        <input type="color" name="color" id="colorPicker">
                </div>
        </div>

        <!--
                Hier wird dynamisch aus dem Localstorage inhalt rein geladen
        -->
        <div id="wrapperStorage" class="lightboxWrapper">
                <div class="lightboxContent" id="wrapperStorageContent">

                </div>
        </div>

        <!--
                Neues Canvas erstellen
        -->
        <div id="newCanvas" class="lightboxWrapper">
                <div class="lightboxContent">
                        <h1>Neue Zeichnung</h1>
                        <form id="formCreateCanvas">
                                <label for="canvasName">Name</label>
                                <input type="text" value="New Canvas" required placeholder="Name"
id="canvasName" name="name"/>
                                <label for="canvasHeight">Breite Canvas</label>
                                <input type="text" value="" required placeholder="Breite in Pixel (Max
1200)" id="canvasWidth" name="width"/>
                                <label for="canvasHeight">Höhe Canvas</label>
                                <input type="text" value="" required placeholder="Höhe  in Pixel (Max
1200)" id="canvasHeight" name="height"/>
                                <label for="fileUpload">Bild hochladen</label>
                                <input type="file" id="fileUpload" name="backgroundImage"/>
                                <button id="buttonCreate">Erstellen</button>
                                <a href="#closeLightBox">Schließen</a>
                        </form>
                </div>
        </div>

        <!--
                Neues Canvas erstellen
        -->
        <div id="newGroup" class="lightboxWrapper">
                <div class="lightboxContent">
                        <h1>Neue Gruppe</h1>
                        <form id="formCreateGroup">
                                <input type="text" placeholder="Gruppen Name" name="groupName"
maxlength="6" required>
```

```
                                        <div class="row">
                                                <button>Erstellen</button>
                                        </div>
                                        <div class="row">
                                                <a href="#closeLightBox">Schließen</a>
                                        </div>
                                </form>
                        </div>
                </div>
                <a id="closeLightbox" href="#closeLightbox"></a>

                <!--
                        Drawer Files
                -->
                <script src="js/drawer/CustomNotifierElement.js"></script>
                <script src="js/drawer/HelpFunctions.js"></script>
                <script src="js/drawer/Communication.js"></script>
                <script src="js/drawer/Storage.js"></script>

                <script src="js/drawer/Interaction.js"></script>
                <script src="js/drawer/Pen.js"></script>
                <script src="js/drawer/Canvas.js"></script>
                <script src="js/drawer/Drawer.js"></script>

                <!--
                        INIT Drawer
                -->
                <script src="js/init.js"></script>
</body>
</html>
```

2. js/init.js

```
function init() {
        var drawing = new Drawing({});
}
window.addEventListener('DOMContentLoaded', init, false);
```

3. js/drawer/Drawer.js

```
"use strict";
/**
 * [Drawing description]
 * Inzialisiert alle Objekte und erstellt Events
 */
function Drawing(_optionsPara) {
        var options = {
                width: 1000,
                height: 600,
                background: "white",
                backgroundImage: null,
                canvasWrapper: "canvasWrapper",
    revertLink: "revert",
    downloadLink: "downloadLink",
    storeLink: "storeLink",
    name: "canvasDrawer",
    groupName : null
        };
  var offset = {};
  options = HelpFunction.merge(options, _optionsPara);
```

```
var canvasManager = new Canvas(options);
var communication = new Communicator();
var storageManager = new Storage();
var notifier = document.getElementsByTagName("x-notifier")[0];
var userNumber = "";
var setUserToHear = null;

// Wird automatisch beim Erstellen von Drawing aufgerufen und initzialisiert alle Events.
        this.init = function() {
    var paint = false;
    var that = this;

    this.offsetWrapper();

    // Überwachen von Veränderungen im CanvasWrapper.
    // Bei Änderungen wird der Offset neu gesetzt
    var target = document.querySelector('#canvasWrapper');
    var observer = new MutationObserver(function(mutations) {
        mutations.forEach(function(mutation) {
            that.offsetWrapper();
        });
    });

    var config = { childList : true };
    observer.observe(target, config);


    /**
     * Initzialisierung der Events
     */
        window.addEventListener('resize', function() {
    that.offsetWrapper();
    }, true);

    /**
     * Kommunikation mit anderen Clients
     */
    Interaction.addMessageListener.apply(notifier, [function(e) {
        var operation = e.detail.operation;

        switch(operation) {
            // Anfrage nach verfügbaren Gruppen
            case "getGroups":
                if(options.groupName != null)
                    communication.sendMessage("setGroup", options.groupName, "");
                break;

            // Antwort auf verfügbare Gruppen
            case "setGroup":
                addGroup(e.detail.message, false);
                break;

            // Einfügen von Zeichenoperation, falls in der gleichen Gruppe
            case "addClick":
                if(e.detail.group == options.groupName){
                    e.detail.message = JSON.parse(e.detail.message);
                    canvasManager.addClickPen(e.detail.message.x, e.detail.message.y, e.detail.message.drag, e.de-
tail.message.strokeStyle, e.detail.message.lineJoin, e.detail.message.lineWidth, e.detail.message.drawingI);
                }
                break;

            // Ein User tritt der Gruppe bei.
            case "joinGroup":
                notifier.setContent("User " + e.detail.from + " tritt bei");
```

```
            // Wenn dies der aktuellen Gruppe entspricht, werden ihm die aktuellen Optionen übermittelt.
            if(e.detail.group == options.groupName)
            {
                var sendOptions = {
                    width: options.width,
                    height: options.height,
                    to: e.detail.from
                };
                communication.sendMessage("setGroupOptions", JSON.stringify(sendOptions), options.group-
Name);
            }
            break;

        // Optionen werden zugesendet
        case "setGroupOptions":
            // e.detail.message wird in ein Json Object gewandelt.
            e.detail.message = JSON.parse(e.detail.message);

            // Wenn die Optionen von CLient angefragt wurden
            if(e.detail.message.to == userNumber && options.groupName == e.detail.group)
            {
                notifier.setContent(e.detail.group + " beigetreten");

                // Nutzer zur Kommunikation wird gesetzt. Dieser übermittelt alle bereits gemachten Interaktionen.
                // Dies wird unten angefragt.
                if(setUserToHear == null)
                    setUserToHear = e.detail.from

                var sendOptions = {
                    to: e.detail.from
                };
                options.width = e.detail.message.width;
                options.height = e.detail.message.height;
                options.backgroundImage = null;

                askSave();
                canvasManager.rebuild(options);

                communication.sendMessage("getClicksDone", JSON.stringify(sendOptions))
            }
            break;

        // Bereits gemachte Klicks werden angefragt
        case "getClicksDone":
            e.detail.message = JSON.parse(e.detail.message);

            // Wenn die Nachricht an diesen Nutzer gerichtet ist
            if(e.detail.message.to == userNumber)
            {
                var sendOptions = {
                    to: e.detail.from
                };

                // Anzahl Click holen
                var clicksCount = canvasManager.getClickCount();
                // Für die Anzahl der Clicks eine Nachricht mit Clickposition und Stift verschicken.
                for (var i = 0; i < clicksCount; i++) {
                    var clickPen = canvasManager.getClickPen(i);
                    clickPen.to = e.detail.from;

                    communication.sendMessage("setClicksDone", JSON.stringify(clickPen));
                };
            }
            break;
```

```
            // Bereits gemachte Klicks erhalten
            case "setClicksDone":
                e.detail.message = JSON.parse(e.detail.message);
                if(e.detail.message.to == userNumber)
                {
                    canvasManager.addClickPen(e.detail.message.x, e.detail.message.y, e.detail.message.drag, e.de-
tail.message.strokeStyle, e.detail.message.lineJoin, e.detail.message.lineWidth, e.detail.message.drawingI);
                }
                break;

            // Die User Number wird vom Server erhalten
            case "setUserNumber":
                userNumber = e.detail.number;
                break;

            // Einen Schritt zurück gehen
            case "revertStep":
                if(e.detail.group == options.groupName)
                    canvasManager.revert();
                break;
        }
    }]);

    // Menü für Mal Operationen wird erstellt. CreateMenu toggelt bei Klick eine Visible Klasse.
    var menuDrawOperations = new HelpFunction.createMenu({
        "openElement": "openDrawOperations",
        "closeElement": "closeDrawOperations",
        "visibleAttachElement": "drawOperationsMenu"
    });

    // Menü für Mal Operationen wird erstellt. CreateMenu toggelt bei Klick eine Visible Klasse.
    var menuMainOptions = new HelpFunction.createMenu({
        "openElement": "openMainOptions",
        "closeElement": "closeMainOptions",
        "visibleAttachElement": "mainOptionsWrapper"
    });

    /**
     * Alle lightboxWrapper Klassen werden mit einem Click oder Touch Listener belegt,
     * um bei offener Lightbox auch durch Berührung des dunklen Feldes die Lightbox zu schließen.
     * Lightbox bei Canvas erstellen oder Gruppe erstellen
     */
    var lightboxWrapper = document.querySelectorAll(".lightboxWrapper");
    for (var i = lightboxWrapper.length - 1; i >= 0; i--) {
        Interaction.addClickListener.apply(lightboxWrapper[i], [HelpFunction.closeLightbox]);
    };


    var storageWrapper = document.getElementById("wrapperStorageContent");
    /**
     * Bei Klick auf loadSettings wird der StorageWrapper mit SettingsDaten befüllt und dargestellt.
     */
    var loadSettingsLink = document.getElementById("loadSettings");
    Interaction.addClickListener.apply(loadSettingsLink, [function(e) {
        storageWrapper.innerHTML = "<h2>Gespeicherte Einstellungen</h2>";
        var settings = storageManager.getAllSettingsKeys();

        for (var i = 0; i < settings.length; i++) {
            var link = document.createElement("a");
            var text = document.createTextNode(settings[i]);
            link.appendChild(text);
            storageWrapper.appendChild(link);

            link.addEventListener("click", function(e) {
                var jsonString = storageManager.loadSettings( e.target.text )
```

```
            var jsonObj = JSON.parse(jsonString);
            options.width = jsonObj.width;
            options.height = jsonObj.height;

            askSave();
            canvasManager.rebuild(options);
            HelpFunction.closeLightbox();
        })
    };
    // Eigentliches Klick Event wird ausgeführt und so die Lightbox dargestellt
    return true;
}]);


/**
 * Bei Klick auf loadImages wird der StorageWrapper mit Localstorage Bilder befüllt und dargestellt.
 */
var loadImageLink = document.getElementById("loadImgLink");
Interaction.addClickListener.apply(loadImageLink, [function(e) {
    var images = storageManager.getAllImageKeys();
    storageWrapper.innerHTML = "<h2>Gespeicherte Bilder</h2>";
    for (var i = 0; i < images.length; i++) {
        var link = document.createElement("a");
        var text = document.createTextNode(images[i]);
        link.appendChild(text);
        storageWrapper.appendChild(link);

        link.addEventListener("click", function(e) {
            that.rebuildWithBackground(storageManager.loadImage( e.target.text ));
            HelpFunction.closeLightbox();
        });
    };
    // Eigentliches Klick Event wird ausgeführt und so die Lightbox dargestellt
    return true;
}]);



// Form Canvas erstellen wird submitted
var formCreateCanvas = document.getElementById("formCreateCanvas");
Interaction.addSubmitListener.apply(formCreateCanvas, [function (e) {
    /**
     * Neues Canvas erstellen
     */
    var fields = HelpFunction.readForm.apply(formCreateCanvas);
    options = HelpFunction.merge(options, fields);

    leaveGroup();
    HelpFunction.closeLightbox();

    askSave();
    // Wenn ein Bild hochgeladen wurde, wird es in den Options gespeichert und als Hintergrund eingebunden.
    if(fields.backgroundImage != null && fields.backgroundImage != "")
    {
        var that = this;
        var input =  document.getElementById("fileUpload");

        var fReader = new FileReader();
        fReader.readAsDataURL(input.files[0]);

        fReader.onloadend = function(event){
            options.backgroundImage = new Image();
            options.backgroundImage.src = event.target.result;

            canvasManager.rebuild(options);
```

```
        }
    }
    else {
        canvasManager.rebuild(options);
    }
}]);


// Form Gruppe erstellen wird submitted
var formCreateGroup = document.getElementById("formCreateGroup");
Interaction.addSubmitListener.apply(formCreateGroup, [function (e) {
    var fields = HelpFunction.readForm.apply(formCreateGroup);
    options = HelpFunction.merge(options, fields);

    communication.sendMessage("setGroup", options.groupName, "");
    addGroup(options.groupName, true);
    HelpFunction.closeLightbox();

    formCreateGroup.reset();

    setUserToHear = null;
}]);

// Gruppe auswählen und beitreten
var groupSelection = document.getElementById("groupSelection");
Interaction.addOnChangeListener.apply(groupSelection, [function (e) {
    if(groupSelection.value == "Keine") {
        leaveGroup();
    }
    else {
        setUserToHear = null;
        options.groupName = groupSelection.value;
        communication.sendMessage("joinGroup", "", options.groupName);
    }

}]);

// Canvas als Bild im Lokalstorage speichern
var saveButton = document.getElementById(options.storeLink);
saveButton.addEventListener('click', function(e) {
    var imgName = prompt("Bitte einen Namen zum Speichern eingeben");
    if (imgName != null) {
        if(imgName != "")
        {
            storageManager.saveImage(imgName, canvasManager.getDataUrl());
            notifier.setContent("Erfolgreich ! Kann nun beim nächsten Aufruf geladen werden.");
        }
        else {
            notifier.setContent("Eingabe leer");
        }
    }
}, true);

// Einstellungen als JSON String in Lokalstorage speichern
var saveSettingsButton = document.getElementById("saveSettings");
saveSettingsButton.addEventListener('click', function(e) {
    var settingsName = prompt("Bitte einen Namen zum Speichern eingeben");
    if (settingsName != null) {
        if(settingsName != "")
        {
            var settings = {
                width: options.width,
                height: options.height
            };
            storageManager.saveSettings(settingsName, JSON.stringify(settings));
```

```
            notifier.setContent("Erfolgreich ! Kann nun beim nächsten Aufruf geladen werden.");
        }
        else {
            notifier.setContent("Eingabe leer");
        }
    }
}, true);


// Download des aktuellen Bildes
var downloadButton = document.getElementById(options.downloadLink);
downloadButton.attributes.download = options.name + ".png";
downloadButton.addEventListener('click', function(e) {
    downloadButton.href = canvasManager.getDataUrl();
}, true);


// Zurück Button anlegen. Letzte Mal Operation wird entfernt.
var revertLink = document.getElementById(options.revertLink);
Interaction.addClickListener.apply(revertLink, [function (e) {
    canvasManager.revert();
    if(options.groupName != null)
        communication.sendMessage("revertStep", "", options.groupName);
}]);


//Canvas Touch und Mouse Events anlegen. Auf dem Wrappen. In diesen werden die Canvas hinzugefügt.
var canvasWrapper = document.getElementById(options.canvasWrapper);

// Mouse oder Touch Move beginnt
Interaction.addMouseDownListener.apply(canvasWrapper, [function (e) {
    if(typeof e.targetTouches != 'undefined') {
        e.x = e.targetTouches[e.targetTouches.length - 1].pageX;
        e.y = e.targetTouches[e.targetTouches.length - 1].pageY;
    }
    canvasManager.addClick(e.x - offset.X , e.y - offset.Y , false);

    paint = true;

    if(options.groupName != null)
    {
        var pen = canvasManager.getPenToSend();
        var message = JSON.stringify(
        {
            x : e.x - offset.X,
            y : e.y - offset.Y,
            drag : false,
            strokeStyle : pen.strokeStyle,
            lineJoin : pen.lineJoin,
            lineWidth : pen.lineWidth,
            drawingI : pen.drawingFunctionsI
        });

        communication.sendMessage(
            "addClick",
            message,
            options.groupName
        );
    }
}]);

// Mouse oder Touchbewegung
Interaction.addMouseMoveListener.apply(canvasWrapper, [function (e) {
    if(paint)
    {
```

```
            if(typeof e.targetTouches != 'undefined') {
                e.x = e.targetTouches[e.targetTouches.length - 1].pageX;
                e.y = e.targetTouches[e.targetTouches.length - 1].pageY;
            }

            canvasManager.addClick(e.x - offset.X, e.y - offset.Y, true);

            if(options.groupName != null)
            {
                var pen = canvasManager.getPenToSend();
                var message = JSON.stringify(
                {
                    x : e.x - offset.X,
                    y : e.y - offset.Y,
                    drag : true,
                    strokeStyle : pen.strokeStyle,
                    lineJoin : pen.lineJoin,
                    lineWidth : pen.lineWidth,
                    drawingI : pen.drawingFunctionsI
                });

                communication.sendMessage(
                    "addClick",
                    message,
                    options.groupName
                );
            }
        }
    }]);

    // Touch- oder Clickbewegung hört auf
    Interaction.addClickListener.apply(canvasWrapper, [function (e) {
        paint = false;
    }]);

    // Mouse oder Touchbewgung verlässt das Element
    Interaction.addMouseLeaveListener.apply(canvasWrapper, [function (e) {
        paint = false;
    }]);
}


this.rebuildWithBackground = function (imgAsDataURL) {
    options.backgroundImage = new Image();
    options.backgroundImage.src = imgAsDataURL;

    options.width = options.backgroundImage.width;
    options.height = options.backgroundImage.height;
    askSave();
    canvasManager.rebuild(options);
}

// Optionen werden zusammengeführt
this.rebuild = function(_optionsPara) {
    console.log("drin");
    askSave();
    options = HelpFunction.merge(options, _optionsPara);
    canvasManager.rebuild(options);
}

// CanvasWrapper wird horizental zentriert. Außerdem wird der Offset aktualisiert
this.offsetWrapper = function() {
    var wrapper = document.getElementById(options.canvasWrapper);

    wrapper.style.marginTop = ((window.innerHeight - options.height) / 2) + "px";
```

```
        offset.Y = wrapper.offsetTop;
        offset.X = wrapper.offsetLeft;
    };

    // Gruppe wird zum Select Input hinzugefügt, wenn Sie nicht bereits vorhanden ist.
    function addGroup(groupName, selected) {
        var testIfAlreadyExists = document.querySelector("#groupSelection option[value='" + groupName + "']");

        if(testIfAlreadyExists == null)
        {
            var option = document.createElement("OPTION");
            option.value = groupName;
            if(selected)
                option.selected = "selected";
            option.appendChild(document.createTextNode(groupName));
            document.querySelector("#groupSelection").appendChild(option);
        }
    }

    // Aktuelle Gruppe wird verlassen
    function leaveGroup() {
        if(options.groupName != null) {
            notifier.setContent("Gruppe verlassen");
        }
        options.groupName = null;
        setUserToHear = null;
    }

    // Fragt ob aktuelle Maloperation gespeichert werden soll
    function askSave () {
        var clicksCount = canvasManager.getClickCount();
        if(clicksCount>0) {
            var r = confirm("Möchten Sie Speichern?");
            if (r == true) {
                var imgName = prompt("Bitte einen Namen zum Speichern eingeben");
                if (imgName != null) {
                    if(imgName != "")
                    {
                        storageManager.saveImage(imgName, canvasManager.getDataUrl());
                        notifier.setContent("Erfolgreich ! Kann nun beim nächsten Aufruf geladen werden.");
                    }
                }
            }
        }
        return;
    }

            return this.init();
}
```

4. js/drawer/Canvas.js

```
"use strict";
/**
 * [Canvas description]
 * In dieser Klasse werden die Canvas Dom Elemente erstellt und die eigentliche Zeichenlogik angespiegelt.
 */
function Canvas(options)
{
```

```
var arrayCanvas = new Array();
var clickX = new Array();
var clickY = new Array();
var clickDrag = new Array();
var pens = new Array();

var penManager = new Pen();

// Initialisiert das 1. Canvas Element.
this.init = function() {
        addCanvas();

   return this;
}

/**
 * [getClickPen description]
 * ClickPostition und aktueller Stift werden entsprechend i zurückgegeben
 * @param  {[int]} i [description]
 * Position des Clicks
 * @return {[object]}   [description]
 * Object mit allen Werten wird zurück gegeben
 */
this.getClickPen = function(i) {
   var response = {};
   var pena = this.getPenToSend(i);

   response.x = clickX[i];
   response.y = clickY[i];
   response.drag = clickDrag[i];
   response.strokeStyle = pena.strokeStyle;
   response.lineJoin = pena.lineJoin;
   response.lineWidth = pena.lineWidth;
   response.drawingI = pena.drawingFunctionsI;
   return response;
}

/**
 * [getClickCount description]
 * Anzahl der Click und Touch Operationen abfragen
 * @return {[int]} [description]
 * Anzahl Clicks
 */
this.getClickCount = function() {
   return clickX.length;
}

/**
 * [setPen description]
 * Setzt Mal Art entsprechend des angegeben Stiftes auf dem angegebenen canvas
 * @param {[int]} penNumber [description]
 * Id des Stiften
 * @param {[int]} canvasId  [description]
 * Canvas, welches den Stift anwenden soll
 */
this.setPen = function(penNumber, canvasId) {
   if(arrayCanvas[canvasId].currentPen == penNumber)
      return;

   var options = penManager.getPen(penNumber);

   arrayCanvas[canvasId].ctx.strokeStyle = options.strokeStyle;
   arrayCanvas[canvasId].ctx.lineJoin = options.lineJoin;
   arrayCanvas[canvasId].ctx.lineWidth =  options.lineWidth;
   arrayCanvas[canvasId].ctx.lineCap = 'round';
```

```
    arrayCanvas[canvasId].currentPen = penNumber;
    arrayCanvas[canvasId].drawFunction = options.drawingFunction;

    if(typeof arrayCanvas[canvasId].drawFunction != "function")
        arrayCanvas[canvasId].drawFunction = penManager.getFunction(options.drawingFunctionsI);
}

/**
 * [rebuild description]
 * Setzt das Objekt zurück und übernimmt die übergebenen Optionen
 * @param {{object}} _optionsPara [description]
 * Optionen siehe Drawer
 */
this.rebuild = function(_optionsPara) {
    clickX.length = 0;
    clickY.length = 0;
    clickDrag.length = 0;
    pens.length = 0;
    options = _optionsPara;

    penManager.rebuild();

    while (arrayCanvas.length != 0)
    {
        deleteCanvas();
    }

    addCanvas();
}


/**
 * [revert description]
 * Letzte Bewegung wird entfernt
 */
this.revert = function() {
    if(clickDrag[clickDrag.length - 1 ] == true)
    {
        while(clickDrag[clickDrag.length - 1] == true) {
            clickX.pop();
            clickY.pop();
            clickDrag.pop();
            pens.pop()
        }
        // 1. Click der Move Bewegung wird nicht als Drag interpretiert
        // Deshalb wird es hier zusätzlich zum Drag entfernt
        clickX.pop();
        clickY.pop();
        clickDrag.pop();
        pens.pop()
    }
    else {
        clickX.pop();
        clickY.pop();
        clickDrag.pop();
        pens.pop()
    }

    //
    while((Math.floor(clickX.length / 500) + 1) < arrayCanvas.length) {
        deleteCanvas();
    }
    this.drawAll();
}
```

```
    /**
     * [addClick description]
     * Letzte Bewegung wird hinzugefügt
     */
    this.addClick = function(x, y, dragging) {
        clickX.push(x);
        clickY.push(y);
        clickDrag.push(dragging);

        pens.push(penManager.getCurrentNumber());

        // Jedes Mal wenn 500 Schritte gezeichnet wurden, wird ein neues Canvas erstellt, um die Rechenoperatinen zu
minimieren.
        if (clickX.length % 500 == 0)
        {
            addCanvas();
        }
        this.drawLast();
    }

    /**
     * [getPenToSend description]
     * Pen ohne Funktion holen
     */
    this.getPenToSend = function(index) {
        var pena
        if(typeof index == "undefined")
            pena = penManager.getPen(pens[pens.length - 1]);
        else
            pena = penManager.getPen(pens[index]);

        delete pena.drawingFunction;
        return pena;
    }

    /**
     * [addClickPen description]
     * Bewegung inklusive Stift hinzufügen
     */
    this.addClickPen = function(x, y, dragging, strokeStyle, lineJoin, lineWidth, drawingFunctionsI) {
        clickX.push(x);
        clickY.push(y);
        clickDrag.push(dragging);

        var oldNumber = penManager.getCurrentNumber();
        pens.push(penManager.addPen(strokeStyle, lineJoin, lineWidth, drawingFunctionsI));

        // Jedes Mal wenn 500 Schritte gezeichnet wurden, wird ein neues Canvas erstellt, um die Rechenoperatinen zu
minimieren.
        if (clickX.length % 500 == 0)
        {
            addCanvas();
        }
        this.drawLast();

        var canvasToDraw = Math.floor(clickX.length/500);

        penManager.setCurrentPen(oldNumber);
    }


    /**
     * [drawAll description]
     * Alle Operationen des aktuellen Canvas Elements werden wiederholt
```

```
 */
this.drawAll = function() {
   var canvasToDraw = Math.floor(clickX.length/500);

   drawDefaults(canvasToDraw);

   for (var i = canvasToDraw * 500; i < clickX.length; i++) {
      this.setPen( pens[i], canvasToDraw);

      arrayCanvas[canvasToDraw].drawFunction(arrayCanvas[canvasToDraw].ctx, clickX, clickY, clickDrag, i);
   };
}

/**
 * [drawLast description]
 * Zuletzt hinzugefügte Bewegung wird gezeichnet
 */
this.drawLast = function() {
   var canvasToDraw = Math.floor(clickX.length/500);
   var i = clickX.length - 1;

   this.setPen( pens[i], canvasToDraw);
   arrayCanvas[canvasToDraw].drawFunction(arrayCanvas[canvasToDraw].ctx, clickX, clickY, clickDrag, i);
}

/**
 * [getDataUrl description]
 * Holt die dataUrl von allen Canvas und malt sie auf einen neuen Canvas Layer.
 * Welches dann ebenfalls über die Methode toDataUrl geholt wird und zurück gegeben wird.
 */
this.getDataUrl = function() {
   var images = [];

   for (var i = 0; i < arrayCanvas.length; i++) {
      images[i] = arrayCanvas[i].canvas.toDataURL('image/png');
   };

   addCanvas();
   var canvas = arrayCanvas[arrayCanvas.length - 1];

   for (var i = 0; i < images.length; i++) {
      var img = new Image();
      img.src = images[i];

      canvas.ctx.drawImage(img, 0, 0);
   };


   var downloadImage = canvas.canvas.toDataURL('image/png');
   return downloadImage;
}

/**
 * [drawDefaults description]
 * Malt ein BackgroundImage falls gesetzt
 * @param  {[int]} i [description]
 * Indes des Canvas Elements

 */
var drawDefaults = function(i) {
         arrayCanvas[i].ctx.clearRect(0, 0, options.width, options.height);

         if(options.backgroundImage != null && options.backgroundImage != "" && i == 0)
   {
      arrayCanvas[i].ctx.drawImage(options.backgroundImage, 0, 0, options.width, options.height);
```

```
        }
    };

    /**
     * [addCanvas description]
     * Canvas Layer wird hinzugefügt
     */
    var addCanvas = function() {
        var i = arrayCanvas.length;
        var canv = document.createElement('canvas');
        canv.id = 'canvas_' + i;

        document.getElementById(options.canvasWrapper).style.width = options.width + "px";
        document.getElementById(options.canvasWrapper).appendChild(canv);
        arrayCanvas[i] = {};
        arrayCanvas[i].canvas = canv;
        arrayCanvas[i].ctx = canv.getContext('2d');
        arrayCanvas[i].currentPen = null;

        canv.width = options.width;
        //canvas.style.width = options.width + "px";
        canv.height = options.height;
        if(i != 0)
            canv.style.marginTop = "-" + options.height + "px";
        //canvas.style.height = options.height + "px";
        drawDefaults(i);
    };

    /**
     * [deleteCanvas description]
     * Canvas Layer wird entfernt
     */
    var deleteCanvas = function() {
        var i = arrayCanvas.length - 1;
        var canvas = arrayCanvas[i].canvas;
        canvas.parentNode.removeChild(canvas);;

        arrayCanvas.pop();
    };

    return this.init();
}
```

5. js/drawer/Pen.js

```
"use strict";
/**
 * [Pen description]
 * Erstellt Dom Elemente für Stifte und legt Klickfunktionen für diese ab.
 * Außerdem speichert es Stifte ab, um später darauf zugreifen zu können.
 */
function Pen() {
    var pens = new Array();
    var color = "#000000";
    var lineArt = "round";
    var width = 15;
    var changePen = false;
    var currentDrawingFunction = 0;
    var currentPen = 0;
    var drawingFunctions = new Array();

    // Stift werden initzialisiert und zur Dom hinzugefügt
    this.init = function() {
```

```
    var colorPicker = document.getElementById("colorPicker");
    var strokePicker = document.getElementById("strokePicker");
    var widthPicker = document.getElementById("widthPicker");

    Interaction.addOnChangeListener.apply(colorPicker, [setColor]);
    Interaction.addOnChangeListener.apply(strokePicker, [setLineArt]);
    Interaction.addOnChangeListener.apply(widthPicker, [setLineWidth]);

    initDrawingFunctions();

    var applyElement = document.getElementById("penSelector");

    for (var i = 0; i < drawingFunctions.length; i++) {
        var ele = document.createElement("a");
        var number = i;
        var textNote = document.createTextNode(drawingFunctions[i].name);
        ele.appendChild(textNote);
        ele.setAttribute("id", "pen"+i);
        ele.setAttribute("href", "#");
        applyElement.appendChild(ele);
        ele.addEventListener("click", function(e) {
            setDrawingFunction(e.target.attributes.id.value);
        });
    };


    pens[currentPen] = {
        strokeStyle : color,
        lineJoin : lineArt,
        lineWidth : width,
        drawingFunction : drawingFunctions[currentDrawingFunction].function,
        drawingFunctionsI : currentDrawingFunction
    };
    HelpFunction.toggleClassName(document.getElementById("pen0"), true, "active");
}

/**
 * [getFunction description]
 * Fragt die Function einer MalOperation ab
 * @param  {[type]} i [description]
 * Index der Maloperation
 * @return {[function]}   [description]
 */
this.getFunction = function(i) {
    return drawingFunctions[i].function;
}

/**
 * [initDrawingFunctions description]
 * Erstellt Mal Operationen
 */
function initDrawingFunctions() {

    drawingFunctions[0] = {};
    drawingFunctions[0].name = "Standart Maler";
    drawingFunctions[0].function = function(ctx, clickX, clickY, clickDrag, i) {
        ctx.beginPath();
        if(clickDrag[i] && i) {
            ctx.moveTo(clickX[i-1], clickY[i-1]);
        }
        else {
            ctx.moveTo(clickX[i] -1, clickY[i]);
        }
        ctx.lineTo(clickX[i], clickY[i]);
        ctx.closePath();
```

```
        ctx.stroke();
    };


    drawingFunctions[1] = {};
    drawingFunctions[1].name = "Random Maler";
    drawingFunctions[1].function = function(ctx, clickX, clickY, clickDrag, i) {

        for (var j = -(ctx.lineWidth / 2); j < (ctx.lineWidth / 2); j+= 4) {
            for (var k = -(ctx.lineWidth / 2); k < (ctx.lineWidth / 2); k+= 4) {
                if (Math.random() > 0.5) {
                    ctx.fillStyle = ['red', 'orange', 'yellow', 'green',
                                'light-blue', 'blue', 'purple'][HelpFunction.getRandomInt(0,6)];
                    ctx.fillRect(clickX[i]+j, clickY[i]+k, 4, 4);
                }
            }
        }
    };


    drawingFunctions[2] = {};
    drawingFunctions[2].name = "Spray";
    drawingFunctions[2].function = function(ctx, clickX, clickY, clickDrag, i) {
        /**
         * Malt 50 zufällige Punkte
         */
        for (var j = 50; j--; ) {
            var radius = ctx.lineWidth/2;
            var offsetX = HelpFunction.getRandomInt(-radius, radius);
            var offsetY = HelpFunction.getRandomInt(-radius, radius);
            ctx.fillStyle = ctx.strokeStyle;
            ctx.fillRect(clickX[i] + offsetX, clickY[i] + offsetY, 1, 1);
        }
    };

    drawingFunctions[3] = {};
    drawingFunctions[3].name = "Radierer";
    drawingFunctions[3].function = function(ctx, clickX, clickY, clickDrag, i) {
        ctx.strokeStyle = "white";
        drawingFunctions[0].function(ctx, clickX, clickY, clickDrag, i);
    };


}

/**
 * [setColor description]
 * Setzt Color des Stiftes
 * @param {[type]} paraColor [description]
 */
function setColor (paraColor) {
        // Wenn durch das Change Event ausgelöst wird, wird der Wert aus dem colorPicker genommen.
        if(typeof paraColor === "object")
                color = colorPicker.value;
    else
        color = paraColor;

    changePen = true;
};

/**
 * [setLineArt description]
 * Setzt Linienart des Stiftes
 * @param {[type]} paraLineArt [description]
 */
```

```
function setLineArt (paraLineArt) {
   // Wenn durch das Change Event ausgelöst wird, wird der Wert aus dem colorPicker genommen.
        if(typeof paraLineArt === "object")
                lineArt = strokePicker.value;
        else
        lineArt = paraLineArt;

   changePen = true;
};

/**
 * [setLineWidth description]
 * Setzt Linien Breite
 * @param {[type]} paraLineWidth [description]
 */
function setLineWidth (paraLineWidth) {
   // Wenn durch das Change Event ausgelöst wird, wird der Wert aus dem colorPicker genommen.
        if(typeof paraLineWidth === "object")
                width = widthPicker.value;
   else
                        width = paraLineWidth;

   changePen = true;
};

/**
 * [setDrawingFunction description]
 * Setzt eine neue Mal Operation mit der entsprechenden id
 * @param {[int]} id [description]
 */
function setDrawingFunction(id) {
   var i = parseInt(id.replace("pen", ""));
   if (currentDrawingFunction != i)
   {
      currentDrawingFunction = i;
      changePen = true;

      var prePen = document.querySelector(".active");
      if(prePen != null)
         HelpFunction.toggleClassName(prePen, false, "active");

      HelpFunction.toggleClassName(document.getElementById(id), true, "active");
   }
}

/**
 * [getPen description]
 * Gibt Stift zurück
 * @param  {[int]} i [description]
 * Index des Stiftes
 * @return {[object]}   [description]
 */
this.getPen = function(i) {
   return pens[i];
};

/**
 * [rebuild description]
 * Pen wird zurückgesetzt
 * @return {[type]} [description]
 */
this.rebuild = function() {
   pens.length = 0;
   currentPen = 0;
```

```
    pens[currentPen] = {
        strokeStyle : setColor({}),
        lineJoin : setLineArt({}),
        lineWidth : setLineWidth({}),
        drawingFunction : drawingFunctions[currentDrawingFunction].function,
        drawingFunctionsI : currentDrawingFunction
    };
}

/**
 * [addPen description]
 * Stift hinzufügen
 * @param {[string]} _strokeStyle      [description]
 * @param {[string]} _lineJoin         [description]
 * @param {[string]} _lineWidth        [description]
 * @param {[string]} _drawingFunctions [description]
 */
this.addPen = function(_strokeStyle, _lineJoin, _lineWidth, _drawingFunctions) {
    currentPen = pens.length;

    pens[currentPen] = {
        strokeStyle : _strokeStyle,
        lineJoin : _lineJoin,
        lineWidth : _lineWidth,
        drawingFunction : drawingFunctions[_drawingFunctions].function,
        drawingFunctionsI : _drawingFunctions
    };

    return currentPen;
}

/**
 * [getCurrentNumber description]
 * Aktuelle Nummer des Stiftes abfragen. Dient dazu nicht jedes Mal ein neues Objekt erstellen zu müssen
 * @return {[int]} [description]
 * index des aktuellen Stiftes
 */
this.getCurrentNumber = function() {
    if (changePen === true)
    {
            currentPen = pens.length;

        pens[currentPen] = {
            strokeStyle : color,
            lineJoin : lineArt,
            lineWidth : width,
            drawingFunction : drawingFunctions[currentDrawingFunction].function,
            drawingFunctionsI : currentDrawingFunction

        };
        changePen = false;
    }
    return currentPen;
};

/**
 * [setCurrentPen description]
 * Setzt aktuellen Stift
 * @param {[int]} i [description]
 *
 */
this.setCurrentPen = function(i) {
    currentPen = i;
}
```

```
    /**
     * [getDrawingFunction description]
     * Drawing Funktion des angegebenen Indexes abfragen
     * @param {[int]} i [description]
     * Index der Funktion
     * @return {[function]}  [description]
     */
    this.getDrawingFunction = function(i) {
        return drawingFunction[i].function;
    }


    return this.init();
}
```

6. js/drawer/Interaction.js

```
"use strict";
/**
 * Namesbereich für Interaktionsfunktionen
 * Diese erstellen ein Callback, welches mit lokalen Variablen arbeiten kann.
 * Die Function werden über call und dem Element aufgerufen.
 */
function Interaction() {

}
(function(){
    this.makeCallback = function(callback) {
        return function(e) {
            callback(e);
            e.preventDefault();
        };
    }

    this.addListener = function (listener, callback) {
        this.addEventListener(listener, Interaction.makeCallback(callback));
    }

    this.addClickListener = function(callback) {
        Interaction.addListener.apply(this, ["mouseup", callback]);
        Interaction.addListener.apply(this, ["touchend", callback]);
    }

    this.addMouseDownListener = function(callback) {
        Interaction.addListener.apply(this, ["mousedown", callback]);
        Interaction.addListener.apply(this, ["touchstart", callback]);
    }

    this.addMouseMoveListener = function(callback) {
        Interaction.addListener.apply(this, ["mousemove", callback]);
        Interaction.addListener.apply(this, ["touchmove", callback]);
    }

    this.addMouseLeaveListener = function(callback) {
        Interaction.addListener.apply(this, ["mouseleave", callback]);
        Interaction.addListener.apply(this, ["touchcancel", callback]);
    }

    this.addMessageListener = function(callback) {
        Interaction.addListener.apply(this, ["newMessage", callback]);
    }

    this.addSubmitListener = function(callback) {
```

```
        Interaction.addListener.apply(this, ["submit", callback]);
    }

    this.addOnChangeListener = function(callback) {
        Interaction.addListener.apply(this, ["change", callback]);
    }
}).call(Interaction);
```

7. js/drawer/Storage.js

```
/**
 * [Storage description]
 * Speichert und liest Einstellungen und Bilder aus dem LocalStorage
 */
function Storage() {
        var imagePrefix = "_image";
        var settingsPrefix = "_settings";


        /**
     * Lädt ein gespeichertes Bild.
     * @param sName.
     */
    this.loadImage = function(imgName) {
        return localStorage.getItem(imgName + imagePrefix);
    };

    /**
     * Lädt gespeicherte Einstellungen
     */
    this.loadSettings = function (setName) {
        return localStorage.getItem(setName + settingsPrefix);
    }


    /**
     * Bild mit Prefix abspeichern
     */
    this.saveImage = function(setName, data) {
        localStorage.setItem(setName + imagePrefix, data);
    }

    /**
     * Einstellungen mit Prefix abspeichern
     */
    this.saveSettings = function(setName, data) {
        localStorage.setItem(setName + settingsPrefix, data);
    }

    /**
     * [getAllImageKeys description]
     * Liest alle gespeicherten Bilder aus
     */
    this.getAllImageKeys = function(){
        var border = imagePrefix.length;
        var imageNames = [];
        for(var key in localStorage){
            if(key.endsWith(imagePrefix)){
                var k = key.substr(0,key.length - border);
                imageNames.push(decodeURI(k));
            }
        }
        return imageNames;
```

```
    };

    /**
     * [getAllSettingsKeys description]
              * Liest alle gespeicherten Einstellungen aus
     */
    this.getAllSettingsKeys = function() {
        var border =settingsPrefix.length;
        var settingsName = [];
        for(var key in localStorage){
            if(key.endsWith(settingsPrefix)){
                var k = key.substr(0,key.length - border);
                settingsName.push(k);
            }
        }
        return settingsName;
    };
}
```

8. js/drawer/Communication.js

```
"use strict";
/**
 * [Communicator description]
 * Ein Object, welches zur Kommunikation von Clients dient.
 * Eigentliche Programmierung der Kommunikation befindet sich in der Klasse Drawer.
 * Nachrichten werden über das Custom Element mit einem Event kommuniziert.
 */
function Communicator()
{
    var ws;
    var notifier = document.getElementsByTagName("x-notifier")[0];
    var kennung = "dada";
    var userNumber = "";
    var url = "borsti1.inf.fh-flensburg.de:8080";

    /**
     * [connect description]
     * Wird beim Erstellen des Objekts automatisch aufgerufen siehe Ende des Scripts
     */
              this.connect = function() {
        var that = this;
                          try {
            // Verbindung wird erstellt
            ws = new WebSocket("ws://" + url);

            /**
             * [onopen description]
             * Wenn die Verbindung zum Server erstellt ist
             */
            ws.onopen = function() {
                notifier.setContent("Verbunden mit Server");

                /**
                 * Bestehende Gruppen abrufen
                 */
                that.sendMessage("getGroups", "", "");
            };
            ws.onmessage = function(e) {
                readData(e.data);
            };
            ws.onclose = function() {
                notifier.setContent("Verbindung beendet");
```

```
        };
    }
    catch(e) {
        //console.log(e.message);
    }
        };

/**
 * [sendMessage description]
 * @param {[string]} operation
 * Operation, welche beim Client ausgeführt werden soll.
 * @param {[string]} message
 * Eigentliche Nachrichteninhalt.
 * @param {[string]} group
 * Gruppe die informiert werden soll. Parameter ist optional
 */
this.sendMessage = function(operation, message, group)
{
    ws.send(this.decodePrivat(operation, message, group));
}


/**
 * [readData description]
 * Die erhalten Daten werden decodet und von Json zurück geparst.
 * Außerdem wird das Notifer Element mit einem Event ausgelöst.
 * @param {[event.data]} data [description]
 * Erhaltene Daten vom Websocket
 */
function readData(data) {
    if (isJson(data)) {
        var jsonObject = JSON.parse(data);

        /**
         * [if description]
         * Prüft ob alle notwendigen Parameter gesetzt sind
         */
        if (typeof jsonObject.url === 'undefined' && typeof jsonObject.kennung != 'undefined' && typeof jsonOb-
ject.message != 'undefined')
        {
            if (isOwnMessage(jsonObject))
            {
                readSecret(jsonObject);

                var event = new CustomEvent(
                    "newMessage",
                    {
                        detail: {
                            operation: jsonObject.operation,
                            message: jsonObject.message,
                            group: jsonObject.group,
                            from: jsonObject.from
                        },
                        cancelable: true
                    }
                );

                notifier.dispatchEvent(event);
            }
        }
    }
    else {
        // Client Number auslesen
        if(data.indexOf("+++Als Client") > -1)
        {
```

```
            userNumber = data.replace("+++Als Client ", "");
            userNumber = userNumber.replace(" verbunden mit " + url, "");
            var event = new CustomEvent(
               "newMessage",
               {
                  detail: {
                     number: userNumber,
                     operation: "setUserNumber"
                  },
                  cancelable: true
               }
            );

            notifier.dispatchEvent(event);
         }

      }
   }

   /**
    * [isJson description]
    * Prüft, ob der übergebene String, zu einem Json geparst werden kann.
    * @param  {[string]}  str [description]
    * Zu prüfender String
    * @return {Boolean}     [description]
    * Gibt true oder False zurück
    */
   function isJson(str) {
      try {
         JSON.parse(str);
      } catch (e) {
         return false;
      }
      return true;
   }

   /**
    * [decodePrivat description]
    * Verschlüsselt eine Nachricht und gibt einen JSON String zurück
    * @param  {[string]} operation [description]
    * Operationsname
    * @param  {[string]} message   [description]
    * Eigentliche Nachricht
    * @param  {[string]} group     [description]
    * Gruppe an die geschickt werden soll
    * @return {[string]}           [description]
    * Verschlüsselter Json String
    */
   this.decodePrivat = function(operation, message, group) {
      var json = {
         "kennung" : decode(kennung),
         "message" : decode(message),
         "operation" : decode(operation),
         "from" : decode(userNumber)
      }
      if(typeof group != "undefined" )
         json.group = decode(group);

      return JSON.stringify(json);
   }

   /**
    * [decode description]
    * Verschlüsselt ein Element. Erhöht den charCode Wert aller Buchstaben um 12
    * @param  {[string]} word [description]
```

```
 * Das zu verschlüsselnde Wort
 * @return {[string]}      [description]
 * Das verschlüsselte Wort
 */
function decode(word) {
    var newMessage = "";
    for(var i = 0; i < word.length; i++)
    {
        newMessage += String.fromCharCode(word.charCodeAt(i) + 12);
    }

    return newMessage;
}

/**
 * [encode description]
 * Entschlüsselt ein Element. Verringert den Int Wert aller Buchstaben um 12
 * @param  {[string]} word [description]
 * Das zu entschlüsselnde Wort
 * @return {[string]}      [description]
 * Das entschlüsselte Wort
 */
function encode(word) {

    var newMessage = "";
    if(typeof word != "undefined") {
        for(var i = 0; i < word.length; i++)
        {
            newMessage += String.fromCharCode(word.charCodeAt(i) - 12);
        }
    }

    return newMessage;
}

/**
 * [readSecret description]
 * Entschlüsselt alle Json Einträge des übergebenen Objects
 * @param  {[object]} data [description]
 * Json Object
 */
function readSecret(data) {
    data.operation = encode(data.operation);
    data.message = encode(data.message);

    if(typeof data.group != undefined) {
        data.group = encode(data.group);
    }

    data.from = encode(data.from);
}

/**
 * [isOwnMessage description]
 * Prüft ob die Kennnug der eigenen entspricht.
 * @param  {[object]}  data [description]
 * Json Object
 * @return {Boolean}      [description]
 * Gibt True oder False zurück
 */
function isOwnMessage(data) {
    if(encode(data.kennung) == kennung)
        return true;

    return false;
```

```
    }

        return this.connect();
}
```

9. js/drawer/HelpFunctions.js

```
"use strict";
/**
 * [HelpFunction description]
 * Namensbereich für Hilfsmethoden
 */
function HelpFunction() {

}

/**
 * [description]
 * Staatische Methoden
 * @return {[type]} [description]
 */
(function() {
        // Schließt die Lightbox
        this.closeLightbox = function(e) {
                if(typeof e === "undefined")
                {
                        document.getElementById("closeLightbox").click();
                }
                else if(e.target.className == "lightboxWrapper") {
                        document.getElementById("closeLightbox").click();
                }
        }


        this.merge = function(defaultObject, overWriteObject) {
                for (var prop in overWriteObject) {
                   defaultObject[prop] = overWriteObject[prop];
                }
                return defaultObject;
        }


        /**
         * [menu description]
         * Erstellt ein Menu mit einem Element zum öffnen und schließen
         * @param  {[type]} options
         * @return {[type]}         [description]
         */
        this.createMenu = function(options) {
                var openElement = document.getElementById(options.openElement);
                var closeElement = document.getElementById(options.closeElement);
                var visibleAttachElement = document.getElementById(options.visibleAttachElement);

                var init =  function() {
                        Interaction.addClickListener.apply(openElement, [open]);
                        Interaction.addClickListener.apply(closeElement, [close]);
                }

                var open = function () {
                        HelpFunction.toggleClassName(visibleAttachElement, true, "visible");
                }
                var close = function () {
                        HelpFunction.toggleClassName(visibleAttachElement, false, "visible");
```

```
                }
                init();
        }

        /**
         * [toggleClassName description]
         * Toggelt einen Klassenname auf einem Element
         * @param {[object]} ele        [description]
         * Dom Element
         * @param {[boolean]} visbility  [description]
         * Ob das Element aus oder eingeblendet werden soll
         * @param {[type]} toggleName [description]
         * Name der Toggle Klasse
         */
        this.toggleClassName = function(ele, visbility, toggleName) {
            var className = ' ' + ele.className + ' ';

            if (visbility == false && ~className.indexOf(' ' + toggleName + ' ') ) {
                ele.className = className.replace(' ' + toggleName + ' ', ' ');
            }
            else if(visbility == true){
                ele.className += ' ' + toggleName;
            }
        }

        /**
         * [getRandomInt description]
         * @param {[int]} min [description]
         * Minimum Wert
         * @param {[int]} max [description]
         * Maximal Wert
         * @return {[int]}     [description]
         * Zufälliger Wert
         */
        this.getRandomInt = function(min, max) {
                return Math.floor(Math.random() * (max - min + 1)) + min;
        }

        /**
         * [readForm description]
         * Wird über call und der Form aufgerufen. Liest alle Felder der Form aus und gibt sie in einem Object
mit einsprechenden Namen der Inputs zurück
         * @return {[object]} [description]
         */
        this.readForm = function () {
    var inputs = this.getElementsByTagName("input");
    var fields = {};
    for (var i = 0; i < inputs.length; i++) {
        fields[inputs[i].name] = inputs[i].value;
    };

    var selects = this.getElementsByTagName("select");
    for (var i = 0; i < selects.length; i++) {
        fields[selects[i].name] = selects[i].value;
    };

    var textareas = this.getElementsByTagName("textarea");
    for (var i = 0; i < textareas.length; i++) {
        fields[textareas[i].name] = textareas[i].value;
    };
    //this.reset();
    return fields;
  }

}).call(HelpFunction);
```

10. js/drawer/CustomNotifierElement.js

```
"use strict";
/**
 * [createCustomElement description]
 * Das Custom Element erstellt ein Div, welches in der Mitte des Bildschirms dargestellt wird,
 * wenn man über die Methode setContent Content hinzufügt. Das Element wird nach 3 Sekunden oder einem Klick
wieder entfernt.
 */
function createCustomElement() {
    // Prototyp-Objekt für x-notifier Element anlegen:
    var notifier = Object.create(HTMLElement.prototype);

    // Dem Prototypen ein Attribut "timer" zufügen:
    Object.defineProperty(notifier, "timer", {
        value: null,
        writable: true,
        enumerable: true
    });

    // Macht das Element sichtbar und blendet es nach 3 Sekunden wieder aus
    notifier.setContent = function (string) {
        this.innerHTML = string;
        this.style.display = "block";
        this.setTimer();
    };

    /**
     * [setTimer description]
     * Timer wird gestellt und nach 3 Sekunden wird das Element ausgeblendet.
     */
    notifier.setTimer = function() {
        var that = this;

        this.timer = window.setTimeout(function() {
            that.hide();
        }, 3000);
    };

    /**
     * [hide description]
     * ELement ausblenden
     */
    notifier.hide = function() {
        this.style.display = "none";
    };

    /**
     * [attachedCallback description]
     * Wird ausgeführt, wenn das Element eingefügt wird.
     */
    notifier.attachedCallback = function () {
        this.setAttribute("style", "position: absolute; top: 50px; left: 50%; width: 270px; margin-left:-150px; display:
none; padding: 15px;");
        this.style.backgroundColor = "blue";

        // Wenn geklick wird es sofort ausgeblendet
        this.addEventListener("click", function (e) {
            clearTimeout(this.timer);
            this.hide();
        });
```

```
    };

    // Neuen Elementtyp registrieren, dabei auf obigen Prototypen verweisen.
    // Es entsteht der Element "Konstruktor"
    var notifierEle = document.registerElement('x-notifier', {prototype: notifier});
    document.body.insertBefore(new notifierEle(), document.querySelector("div"));
}

window.addEventListener("DOMContentLoaded", createCustomElement);
```

10. css/style.css

```
/**
 * Reset CSS
 * Um unterschiedliche Darstellungen zu vermeiden
 */
* {
        margin: 0;
        padding: 0;
}
/**
 * Elemente haben feste Breite. Ohne zusätzlich Padding oder Border
 */
html {
  box-sizing: border-box;
}
*, *:before, *:after {
  box-sizing: inherit;
}


/**
 * Standarts definieren
 */
body {
        background: #a0a0a0;
}
input {
        display: block;
}
ul > li {
        list-style: none;
}


/**
 * Farbe für Link Icons
 */
a.iconLine, a.iconLine:visited {
        color: black;
}

/**
 * Optionen Menü
 */
.mainOptionsWrapper.visible #closeMainOptions{
        display: block;
}
.mainOptionsWrapper.visible #openMainOptions{
        display: none;
}
.mainOptionsWrapper #closeMainOptions{
        display: none;
```

```
}
.mainOptionsWrapper #openMainOptions{
	display: block;
}

.mainOptionsWrapper {
	position: fixed;
	top:0px;
	left: 0px;
}
.mainOptionsWrapper a {
	margin-left: 10px;
	margin-top: 10px;
}

.mainOptionsWrapper.visible .mainOptionsMenu {
	display: block;
}
.mainOptionsWrapper .mainOptionsMenu {
	display: none;
	background-color: white;
	padding: 15px;
}


/**
 * Mal Operationen Menu
 */
.drawOperations {
	position: fixed;
	right: 0;
	top:0px;
	width: 290px;
	margin-right: -250px;
}
.drawOperations.visible {
	margin-right: 0;
}
.openCloseMenu {
	padding-top: 15px;
	width:40px;
}
.paintOptions {
	width: 250px;
	background-color: white;
	padding: 15px;
}
.drawOperations #closeDrawOperations{
	display: none;
}
.drawOperations #openDrawOperations{
	display: block;
}
.drawOperations.visible #openDrawOperations{
	display:none;
}
.drawOperations.visible #closeDrawOperations{
	display: block;
}


.paintOptions a.active{
	font-weight: bold;
}
.paintOptions .penSelectorLabel, .paintOptions label, .paintOptions #revert {
```

```
		margin-top: 20px;
		display: block;
}
.paintOptions *:first-child {
		margin-top: 0;
}
.paintOptions #penSelector a{
		display: block;
}

/**
 * Some helping Classes
 */
.floating {
		float: left;
}
.clearAfter:after {
		content: ".";
    clear: both;
    display: block;
    visibility: hidden;
    height: 0px;
}


/**
 * Canvas
 */
canvas {
		display: block;
		margin: 0 auto;
}

#canvasWrapper {
		margin: 0 auto;
}
#canvasWrapper canvas:first-child{
		background: white;
}

/**
 * Link Default Style
 */
button {
}
a, a:visited {

}
a:hover {

}


/**
 * Click Events
 */
#newGroup:target {
		display:block;
}
#newCanvas:target {
		display:block;
}
#wrapperStorage:target {
		display:block;
}
```

```
/**
 * Lightbox Klassen
 */
.lightboxWrapper {
	display: none;
	position: absolute;
	top: 0;
	bottom: 0;
	height: 100vh;
	width: 100%;

	background-color: rgba(0, 0, 0, 0.7);
}
.lightboxWrapper .lightboxContent {
	position: fixed;
	top: 50%;
	left: 50%;
	height: 400px;
	width: 400px;
	margin-top: -200px;
	margin-left: -200px;
	padding: 20px;

	background: white;
}
.lightboxWrapper .lightboxContent *{
	display: block;
}
.lightboxWrapper .lightboxContent input {
	margin-bottom: 15px;
}
#closeLightbox {
	display: none;
}
```