Projektdokumentation

Entwicklung eines Radios als Webapplication

Dokumentation zur Hausarbeit für das
Modul Webtechnologien an der Hochschule Flensburg

von **Sven Kuhlmann**

MatrikelNr: 610292
Student: Sven Kuhlmann
Geboren am: 09.01.1994 in Leer
Durchführungszeitraum: 01.05.2017 – 15.06.2017

INHALTSVERZEICHNIS

1. EINFÜHRUNG

Fachbegriffe werden in keinem Glossar genauer beschrieben, weshalb diese und verwendeten Abkürzungen in den Fußnoten ausgeschrieben werden.

Quellcode wird logisch auf das wesentliche begrenzt und reduziert. Dies ist mit „…“ in der gesamten Dokumentation angegeben. Ist der Schnitt eindeutig logisch, so ist dieser nicht angegeben.

Die Informationen richten sich selbstverständlich an Frauen und Männer gleichermaßen. Im Text wurde jedoch zugunsten der Lesbarkeit und aus sprachlichen Gründen nur die männliche Form gewählt.

2. LAYOUT

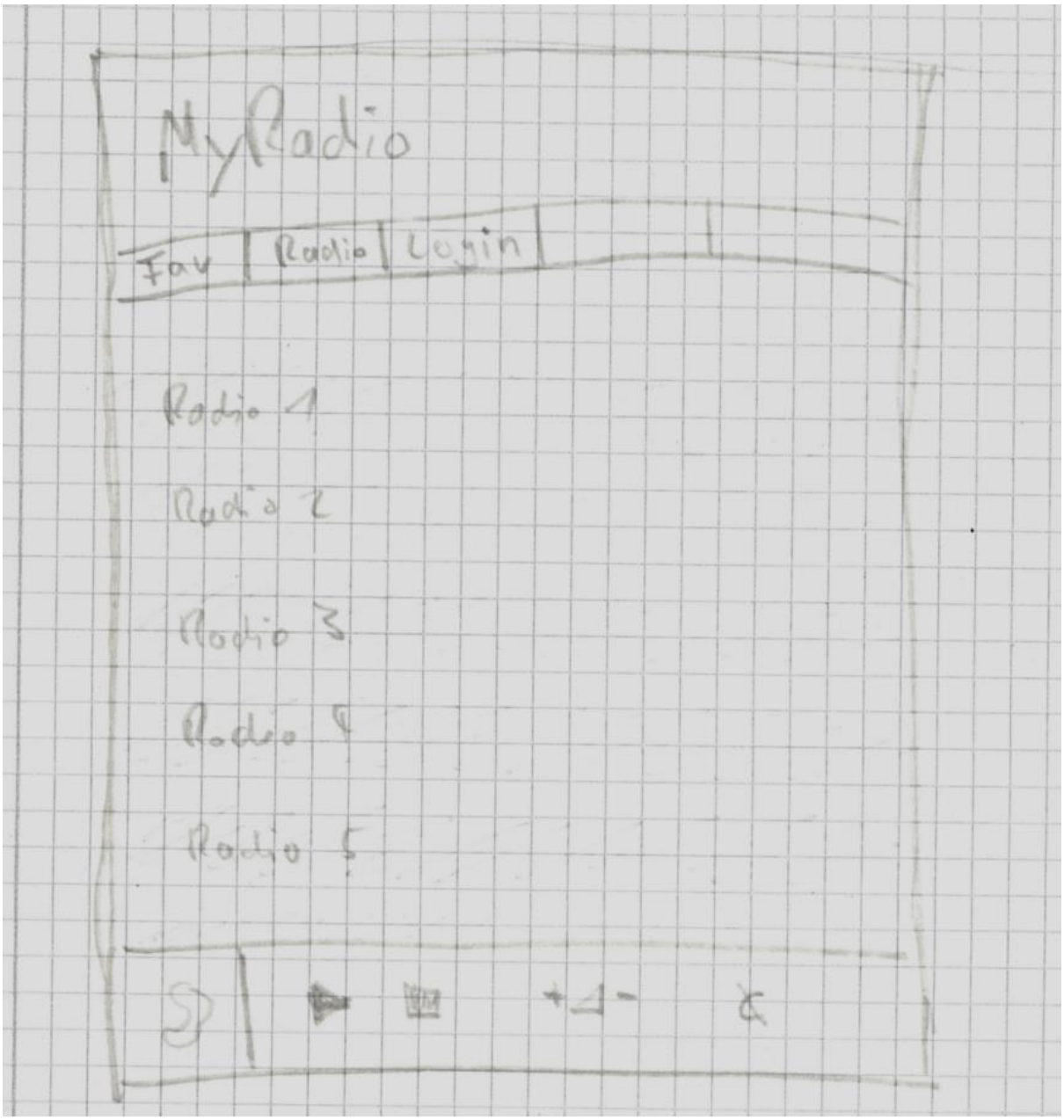
2.1. ENTWURF

Zu Beginn des Projektes wurde in der Vorbereitung von mir eine Skizze zum Layout erstellt. Durch diese Skizze wird schon vor der Programmierung überlegt, in welche Richtung die Software gehen soll. Dadurch werden sich Gedanken gemacht und man hat damit bereits eine Vorstellung von der Umsetzung. Anhand der Skizze habe ich mir einen fixen Header[1] mit Navigation vorgestellt. Dieses harmoniert sehr gut mit der fixen Playbar[2] die im Footer[3] das Audio-Element enthält und den aktuellen Stream anzeigt. Zwischen diesen beiden Bereichen sollen die Radio-Elemente, die abgespielt werden können, angezeigt werden. Die Playbar beinhaltet hier einmal das Bild des abzuspielenden Streams und die Standard Steuerungselementen wie Start/Pause und Stumm schalten. (Siehe folgendes Skizze)

[1] Abschnitt im oberen Bereich der Seite

[2] Eine Leiste mit Steuerungselementen

[3] Abschnitt im unteren Bereich der Seite



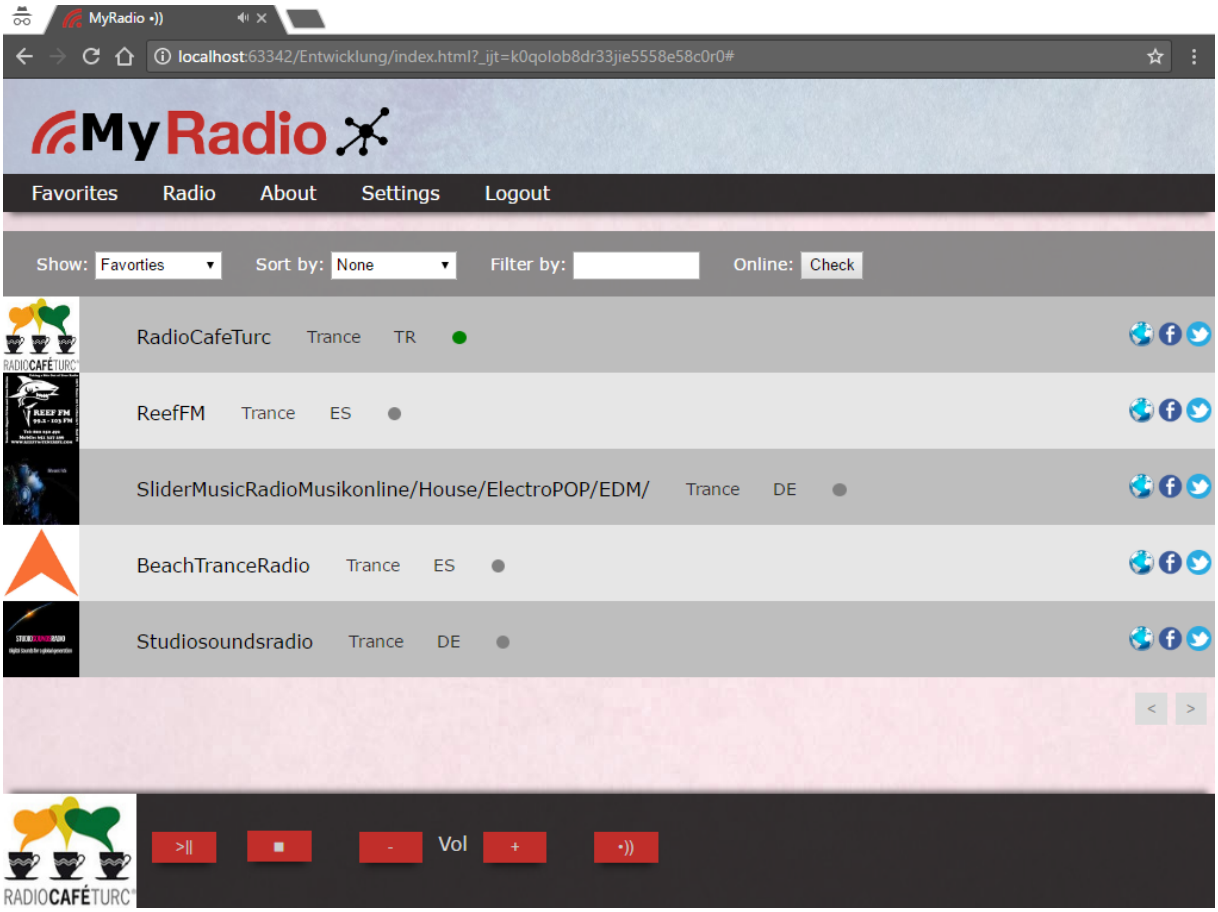
Entwurfsskizze

2.2. UMSETZUNG

Die Umsetzung des Layouts erfolgte sehr nah an dem Entwurf. Lediglich an der Navigation hat sich etwas geändert. Außerdem ist eine zusätzliche Leiste zum Filtern[4] und Sortieren hinzugekommen.

Die Navigation wurde um ein paar Elemente, zur besseren Bedienung, ergänzt. (Siehe folgendes Bild)

[4] Begrenzen der anzuzeigenden Elemente



Umgesetztes Layout von MyRadio

Alle Bereiche wurden hierbei logisch und sinnvoll platziert. Es wird dabei viel Wert auf die Erfahrungen der Nutzer gelegt. Durch die gewählte Platzierung, die auch in der Art auf anderen Seiten zu finden ist, kann sich der Nutzer leicht zurechtfinden. Der Nutzer weiß bereits von andern Seite, wo die Navigation und andere Bereiche aufzufinden sind. Durch eine korrekte Bezeichnung der Navigations-Elementen kann er somit auch vermuten, was sich hinter diesen Punkten verbirgt.
Der Filter wurde so bezeichnet, dass auch hier keine Missverständnisse auftreten können.

Das Abspielen der Elemente erfolgt durch simples anklicken dieser, die untereinander angeordnet sind.
Alle Farben, die in den Settings durch das Layout geändert werden können, stehen in einem angenehmen Kontrast und Verhältnis. Durch das Testen mit einigen Nutzern, konnten diese als sehr angenehm beurteilt werden.

Das Layout wurde in der Struktur mit HTML5 und vom Design mit CSS3 umgesetzt. Die erzeugten Elemente bekommen ihre Designzuordnung über CSS-Klassen. Die Elemente innerhalb der mittleren Sektion werden in einem Float-Layout[5] angeordnet.

[5] Fließend angeordnetes Layout

Die HTML-Datei beinhaltet im Head meta-tags, die Imports von Scripte und Style, das Favicon und ein Titel. Im Body sind innerhalb eines „wrappers[6]“ der Header für das Logo, ein umgeformte Menü-Checkbox (mittels CSS) für das Responsive Layout, die Navigation, den Content innerhalb der Section und einen Footer. (Siehe folgenden Code)

```
<!DOCTYPE HTML>
<html>
   <head>
      <title>MyRadio &bull;))</title>
        <link rel="icon" href="images/favicon.png">
        <!-- Meta -->
        <meta content="text/html; charset=utf-8" http-equiv="Content-Type">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <!-- JS -->
        <script src="js/script.js"></script>
        <script src="js/classes.js"></script>
        <!-- CSS -->
        <link rel="stylesheet" href="css/style.css">
   </head>
   <body>
        <div id="wrapper" class="wrapper">
            <header>
                <a href="#"><img id="logo" alt="Logo" src="images/logo.png" class="logo"/></a>
            </header>
            <input type="checkbox" id="menu" class="menu"/>
            <label class="menuLabel" for="menu"></label>
            <nav>
                <ul id="nav">
                    <li id="favorites"><a href="#">Favorites</a></li>
                    <li id="radio"><a href="#">Radio</a></li>
                    <li id="about"><a href="#">About</a></li>
                    <li id="settings"><a href="#">Settings</a></li>
                    <li id="login"><a href="#">Login</a></li>
                </ul>
            </nav>
            <section>
                <!-- content -->
                <h2>Empty content</h2>
            </section>
            <footer>
                <div id="info"><p>&copy;2017 Diese Seite wurde von Sven Kuhlmann im Rahmen einer
                    Hausarbeit für das Modul Webtechnologien erstellt.</p></div>
            </footer>
        </div>
   </body>
</html>
```

HTML-Datei

2.3. BEDIENKONZEPT

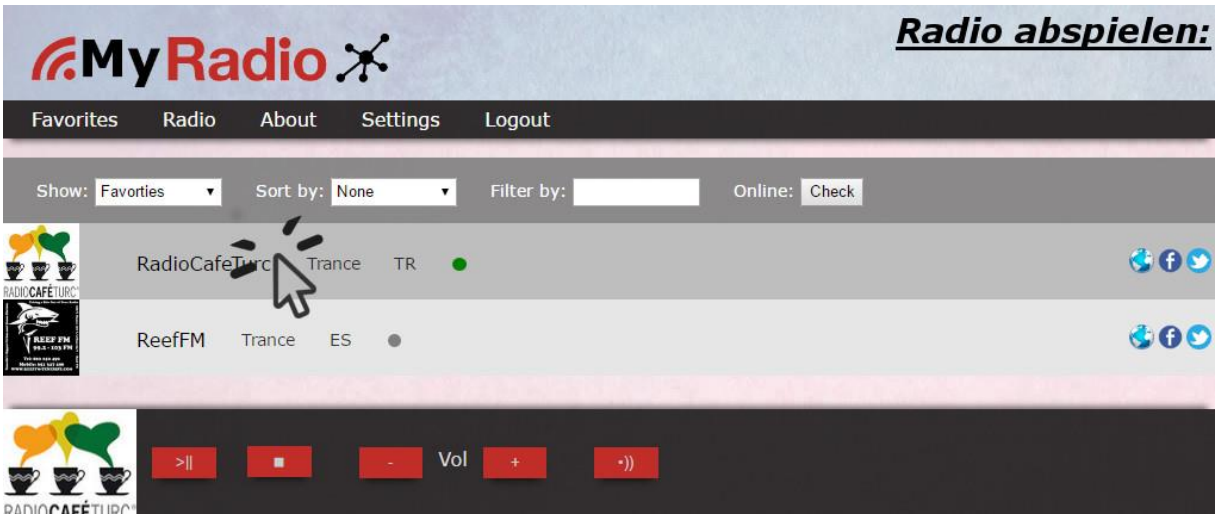
Vorweg: Zum Bedienkonzept kann auch ein *kleines* Tutorial auf der Seite unter dem Punkt „About“ gefunden werden.

Beim Laden der Seite wird, wenn kein Login stattgefunden hat, immer der Login durch aufrufen dessen erzwungen. Der Login für die Webseite wird über localSession in einer Session gespeichert und somit ein Reload der Seite mit einem automatischen Login verbunden.

[6] Ein Container der Elemente umgibt

Durch eine Synchronisation des localStorage (Login-Daten und Favoriten) und localSession (nur aktueller Stream) wird die Möglichkeit, sich an einem anderen Client an zu melden, sichergestellt. Zudem wird nach der Anmeldung der Stream der auf einem anderen Client unter diesem Login läuft, automatisch abgespielt.

Das Bedienkonzept richtet sich an ähnlichen Konzepten von anderen Streaming-Seiten. Eine List der abzuspielenden Elemente ist in der Mitte der Seite zu finden. Durch einfaches anklicken dieser Elemente (Siehe folgendes Bild) lässt sich der Stream für das Radio starten. Dabei wird die Playbar eingeblendet. Diese ist standardmäßig ausgeblendet und zeigt sich nur bei einem aktiven Stream, um hier eine schöne Harmonie zu gewährleisten.

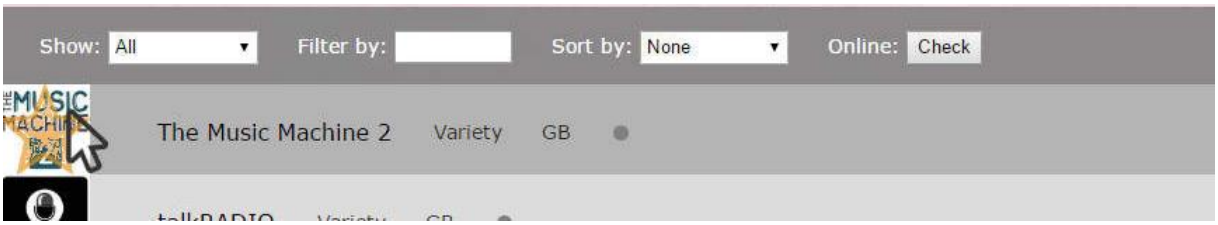


Stream abspielen

Ebenso lassen sich durch das anklicken die Bedienelemente in der Playbar bedienen. Diese sind bei dem Pausieren oder Stumm schalten hervorgehoben, um dieses den Nutzer sichtbar zu machen.

Der Filter-Leiste beinhaltet gewohnte Dropdown Menüs zum Reduzieren der Elemente oder dem Sortieren dieser. Der Filter wird mit dem betätigen der Entertaste ausgelöst oder bei leerem Listenitem zurückgesetzt.

Das Hinzufügen eines Favoriten zu der Favoritenliste geschieht über das Hovern und Anklicken des Bildes von einem Stream. (Siehe folgendes Bild)



Favoriten ergänzen

Durch das klicken auf die Feile unterhalb der Radio-Liste lässt sich hier, wenn möglich, durch die Seiten durchblättern.

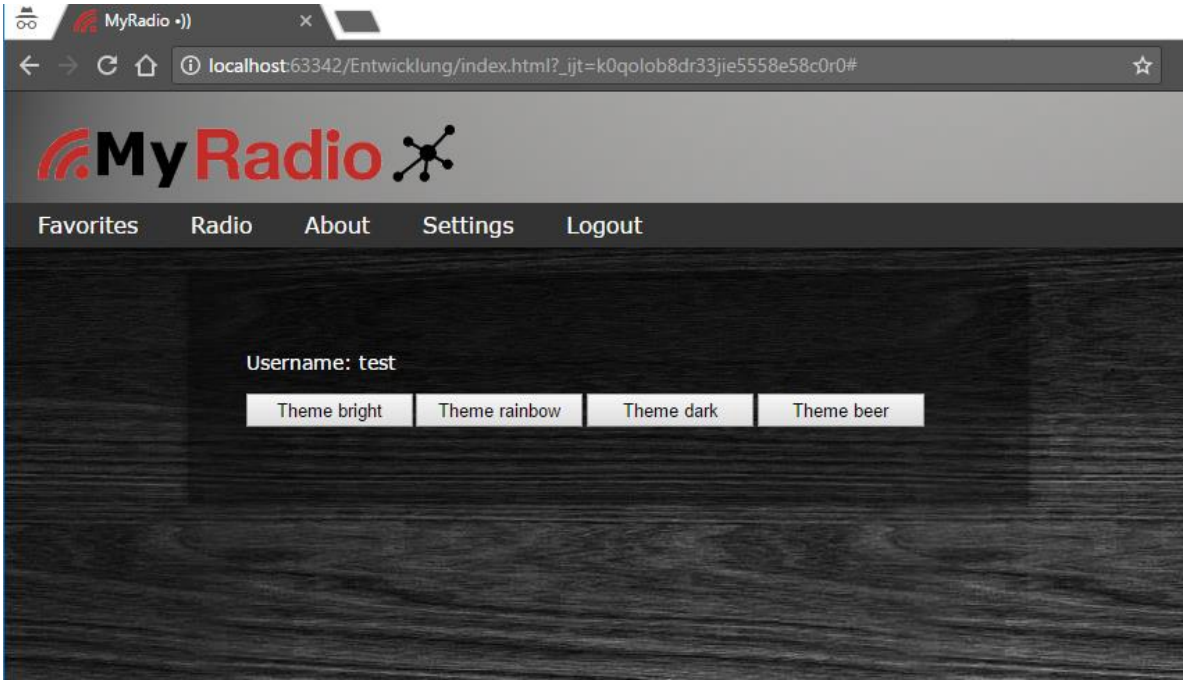
2.3. THEME

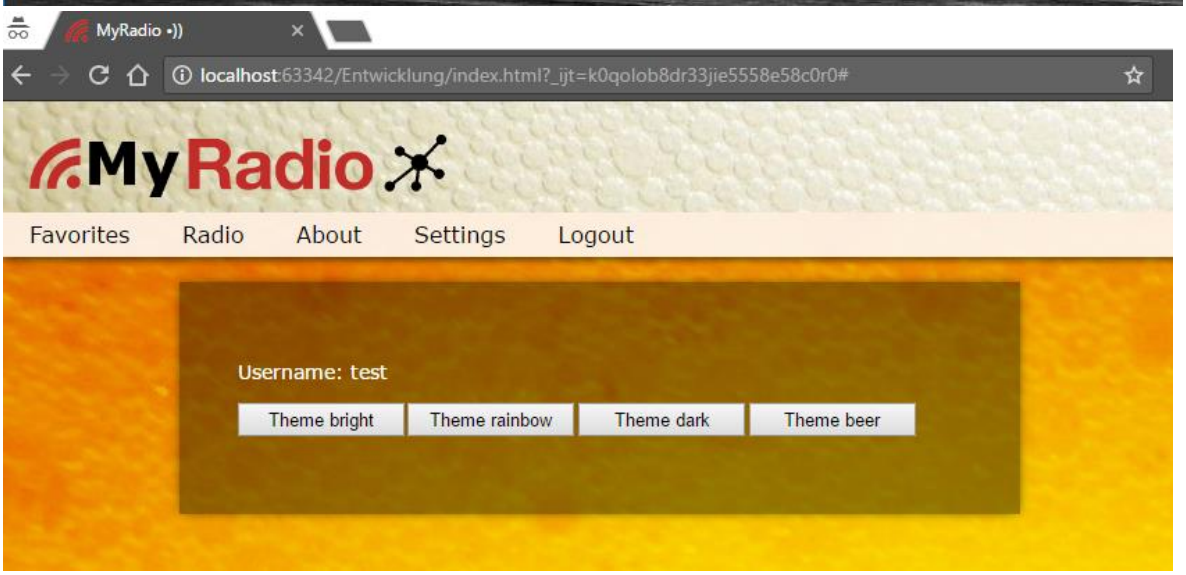
Der Benutzer ist nach einem erfolgreichen Login in der Lage, unter dem dann auftauchenden Reiter *Settings* seinen Theme zu ändern. Diese werden in den Benutzerdaten im localStorage (dazu unten mehr) gespeichert.

Zudem werden diese auch Synchronisiert, wodurch der Benutzer überall sein Layout wiederfindet.
Dies gilt nur wenn ein anderer Client angemeldet ist, mit dem er Synchronisieren kann.

Folgende Themes stehen zur verfügung: Bright(Default), Rainbow, Dark und Beer

Setting-Menü (Theme wahl):





3. LISTENDATEN

3.1. ERZEUGUNG

Die Liste besteht aus Dateneinträgen, die Custom-Elements[7] sind. Diese sind mit radio-entry benannten und sehen generiert wie folgt aus:

Radio-Filter ->

Radio-Eintrag ->

Kind-Elemente vom Radio-Eintrag ->

```
▼<div class="radio-wrp" id="radioList">
  ▶<list-filter class="filter" id="filter-bar">…</list
  ▶<radio-entry class="radio-elmnt" id="57028" style="
  ▶<radio-entry class="radio-elmnt" id="56979" style="
  ▶<radio-entry class="radio-elmnt" id="56936" style="
  ▶<radio-entry class="radio-elmnt" id="56930" style="
  ▼<radio-entry class="radio-elmnt" id="56915" style="
    <img class="radio-img" alt="Radio image" src="htt
    4bde-8fe4-9fc541259cb2.jpg">
    <img class="radio-fav" alt="Add Fav" src="images/
    <fieldset class="radio-text radio-title">Studioso
    <fieldset class="radio-text">Trance</fieldset>
    <fieldset class="radio-text">DE</fieldset>
    <img class="radio-state" alt="State" src="images/
    ▶<div class="radio-icon">…</div>
    ▶<div class="radio-icon">…</div>
    ▶<div class="radio-icon">…</div>
    <div class="radio-click-wrp"></div>
  </radio-entry>
```

Radio-Liste (Abgeschnitten)

Die Radio-Einträge sind, wie die anderen Custom-Elements auch, am anfang der JavaScript-Datei definiert. Für die definition wird eine Referenz auf die Klasse, aus der die Objecte erstellt werden, gesetzt:

```
customElements.define('radio-entry',RadioElement);
```

Für die Erzeugung der Radio-Einträge wird innerhalb einer Schleife, nach dem hinzufügen des Filters, eine Instanz von der Klasse Radio-Element erstellt:

```
…
wrapper.appendChild(new RadioFilter('filter-bar'));
let counter = 0;
let elements = [];
for(let radio in jsonRadio){
    elements[counter] = new RadioElement(jsonRadio[counter]);
    wrapper.appendChild(elements[counter]);
    counter++;
}
…
```

Diese werden dann einem „wrapper“ und darauf folgend der section auf DOM[8] basis hinzugefügt.

[7] Eigen erstellte HTML-Elemente

[8] Document Object Model, ein Repräsentant des Dokuments

4. CUSTOM-ELEMENTS

4.1. RADIO-EINTRAG

Bei der erzeugung einer Instanz wird im connectedCallback das element durch das aufrufen entsprechender Methoden zusammen gebaut:

```
connectedCallback() {
    this.className += this._class;
    this.setAttribute('id', this._id);
    this.style.backgroundColor =
        `rgb(${this._color.r}, ${this._color.g}, ${this._color.b})`;
    // Create and set all attrubutes -----------
    let thumbImage = this.createImage();
    this.appendChild(thumbImage);
    this.appendChild(this.createFav(thumbImage));
    this.appendChild(this.createTag('name'));
    this.appendChild(this.createTag('genre'));
    this.appendChild(this.createTag('country'));
    this.appendChild(this.createState());
…
```

Jedes Element enthält einen Event-Klick-Listener[9], welcher bei der Registrierung eines Klicks die Methode startPlay(item) von dem Object myRadio aufruft. Diese Methode versucht den Stream von dem Item zu starten, indem er ihn dem Audio-Tag hinzufügt. Zudem blendet sie die Playbar ein.

Folgende Features zum Anzeigen besitzt ein Radio-Element:

Thumbnail	Name	Genre
Land	Online-Status (Wenn geprüft)	Sozial-Media links

Ist ein Thumbnail nicht verfügbar, so wird ein vorgefertigtes, welches darauf hinweist, angezeigt.
Sozia-Media links werden in einer Schleife dynamisch nach Verfügbarkeit generiert.
Der Online-Status kann über die Filter-Liste geprüft werden.

[9] Eine Funktion die bei einem Event eine Funktion aufruft

4.2. RADIO-FILTER

Auch hier wird im connectedCallback das Element zusammengebaut:

```
…
connectedCallback() {
    // Create the filter
    this.className += this._class;
    this.setAttribute('id', this._id);
    let wrapper = document.createElement('div');
    wrapper.className += 'filter-wrp';
    // Create control elements and edd them to the wrapper
    wrapper.appendChild(this.createShow());
    wrapper.appendChild(this.createSort());
    wrapper.appendChild(this.createFilter());
    wrapper.appendChild(this.createOnlineCheck());
    this.appendChild(wrapper);
…
```

Filter:

Das Input-Feld zum Filter in der Filter-Liste ist ein HTML5 <input> vom Typ *search*. Das Filtern funktioniert, indem bei jeder Erstellung der aktuellen Radio-Liste, diese in Form eines Arrays in „myRadio.radioList“ gespeichert wird. Bei der Filter-Funktion wird nun innerhalb dieser Liste nach dem Begriff im Input-Feld gesucht. Löscht der Nutzer diesen Filter wieder, wird die alte Liste, die in „RadioElement.oldList“ zwischengespeichert ist, geladen. Dadurch bekommt der die alte Liste zurück, obwohl sich die aktuelle Liste aktualisiert hat.

Sortierung:

Es gibt für jedes verarbeitete Attribut eines Radio-Elements eine Sortiermöglichkeit. Diese unterscheiden sich nochmal zwischen DESC (absteigend) und ASC (aufsteigend). Für die Sortierung wird in einer Funktion die aktuelle Liste in „myRadio.radioList“ sortiert und dann mit der Funktion „myRadio.createRadioList(list)“ die angezeigte Liste neu erstellt.
Als sortierverfahren wird *Bubblesort* mithilfe der Funktion „obj.sort(function)“ für das jeweilige Attribut verarbeitet.

Anzeige (Show):

Diese Auswahl ermöglicht es für jede Option unterschiedliche Datenstände vom Dirble-Server zu laden. Durch das anklicken einer Option wird ein jeweils anderer Link + Parameter in die „myRadio.dirblePage“und „myRadio.dirbleRequest“ geladen. Danach wird die Funktion „myRadio.loadDirbleConnection()“ aufgerufen, welche die Daten anhand der oben genannten Attribute von Server holt und anschließend die Radio-Liste erstellt.

Online check:

Hier wird die Funktion „myRadio.checkOnlineState()“ aufgerufen. Diese versucht für den Stream von jedem List-Item in der aktuellen Radio-List über „myRadio.radioList“ einen Audio-Stream zu starten. Durch ein erstelltes Audio-Tag „let audio = new Audio()“ wird ein lautloser *.load()* versuch getätigt. Ist dieser erfolgreich oder nicht, wird das Bild zur Anzeige der Verfügbarkeit mit der Funktion „myRadio.changeOnlineState(element, status)“ geändert.

4.3. PLAY-BAR

In der Playbar werden mithilfe von Hilfsfunktionen bei der Erzeugung alle Attribute zusammengestellt:

```
…
connectedCallback() {
    // Create the audio-item
    this._audio = PlayBar.createAudio();
    // Create the bar
    this.className += this._class;
    this.setAttribute('id', this._id);
    let wrapper = document.createElement('div');
    wrapper.className += 'bar-wrp';
    this.appendChild(PlayBar.createImage());
    // Create control elements and edd them to the wrapper
    wrapper.appendChild(this.createPlayHalt());
    wrapper.appendChild(this.createStop());
    wrapper.appendChild(this.createVol('down'));
    wrapper.appendChild(document.createTextNode('Vol'));
    wrapper.appendChild(this.createVol('up'));
    wrapper.appendChild(this.createMute());
    wrapper.appendChild(this._audio);
    this.appendChild(wrapper);
…
```

Diese wird bei idem Laden der Seite einmalig erstellt, anders als die anderen Klassen/Objekte, und dann nach Bedarf ausgeblendet.

4.4. LOGIN

Ein Objekt vom Login wird immer dann erzeugt, wenn beim Aufruf der Seite kein valider Benutzer angemeldet ist. Diese Instanz wird nach einer Login-Prüfung von der Funktion „myRadio.checkLogin(next)“ erstellt. Diese Funktion spielt in der ganzen Webapplikation die **größte** Rolle. Alle Seitenaufrufe und Nachlad-Requests (AJAX[10]) werden über diese Methode getätigt, da sie den Login prüft und eine Menge anderer Hilfsfunktionen ausführt. Somit sollte dieses bei Erweiterung der Applikation auch beachtet werden. Die Funktion bekommt einen Wert namens *Next* mitgegeben, welcher zur Identifizierung dient, was nach der Prüfung gemacht werden soll (welche Funktion als nächstes geladen werden soll). Dies ist somit eine **wichtige Schnittstelle.**

„CheckLogin“ prüft dabei erst im localSession nach einer aktiven Sitzung und führt andernfalls ein Login aus.

Bei der Registrierung wird im localStorage ein User hinzugefügt. Dabei werden folgende Daten gespeichert:

TimeStamp	ID	Name	Password (Gehasht)

Der TimeStamp dient zum Datenabgleich. Die ID ist ein zufällig generierte, eindeutige 8-Ställige Nummer.
Der Name wird aus dem Login-Fenster übernommen. Das Password wird bei der Registrierung mit einer eigen

[10] Asynchronous JavaScript and XML, dynamisches nachladen von Daten

geschriebenen Hash[11]-Funktion „login.hash(string)“ gehasht. Dabei wird lediglich der Hash-Code gespeichert, **um das Arbeiten mit Passwörtern im Klartext zu umgehen!**
Beim Login versuch über „login.loginUser(name,password)“ werden somit nur die Hash-Codes verglichen.

5. AUDIO-ELEMENT

Die Playbar enthält hierbei den einzigen Audio-Tag im DOM. Dieser dient zum Abspielen des Streams und dessen Steuerung. Innerhalb des Objektes wird mithilfe eines „audio“-Attributes die Modifikation des Stream gehandelt. Beim Starten eines Streams wird die Funktion „myRadio.startPlay(item)“ und dadurch die Funktion „showPlayBar(valid)“ aufgerufen. Letzteres aktualisiert ebenfalls das Bild und andere Meta-Daten des Audio-Tags in diesm Objekt.

Es werden alle vorgegeben Funktion des Audio-Tags, wie üblich, genutzt. Mittels *audio.pause()* wird so zum Beispiel der Stream pausiert. Allein die Stop-Funktion unterscheidet sich. Hier wird der Stream pausiert und die Source durch den Code:

```
data:audio/wav;base64,UklGRiQAAABXQVZFZm10IBAAAAABAAEAVFYAAFRWAAABAAgAZGF0YQAAAAA=
```

auf einen neutralen ausgangs Status gebracht (clean state).

6. WEBSOCKET

6.1. NACHRICHTENKOMMUNIKATION

Daten werden ausschließlich als JSON verarbeitet. Für die Nachrichtenkommunikation wird bei dem laden der Seite eine Websocket-Verbindung mit der Funktion „sync.createConnection“ aufgebaut. Diese lauscht durchgehen auf Daten. Alle Daten die nicht den gleichen call-sign[12] (auch sign) verwenden, werden verworfen.

Die Umsetzung lässt sich in zwei Teile zerlegen: Request und Response.

Request

Bei dem Laden der Seite wird immer ein Request mittels „sync.request“ über den Broadcast-Server verteilt. Dieser Request wird mit dem String „+++GibData+++“ als Message versendet.

Response

Alle Clients lauschen auf ein Request um diesen über „sync.response“ zu beantworten. Hier wird bei einem Request eine Message mit allen Benutzerdaten und Favoriten sowie Stream-Session zusammengestellt. Diese wird dann übertragen und von dem Requester verarbeitet.

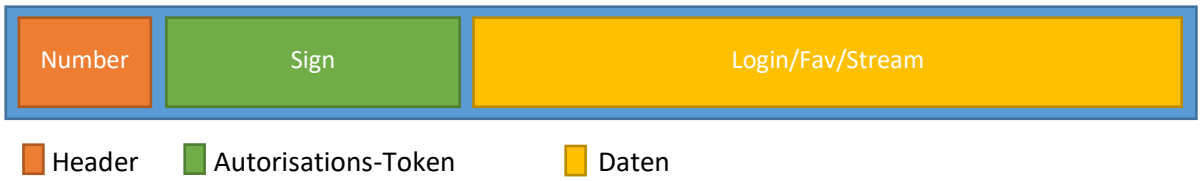
[11] Streufunktion für eine Prüfsumme

[12] Rufzeichen zur Identifizierung gleicher Clients

6.2. ZERLEGUNG

Vor jedem Aufruf von „sync.send“ wird mit „sync.checkDataSplit“ überprüft, ob der Datenstring eine maximale Länge von 500 überschreitet. Ist dies der Fall wird der String zerlegt und der Daten-Header unter „number“ nummeriert. Hier ist 0 immer der Hinweis auf das letzte Paket. Am anderen Client werden diese Datenpakete vor dem Verarbeiten wieder zusammengesetzt, wenn die „number“ größer 1 ist.

Aufbau der Pakete:



6.3 JSON-BEISPIELE

Verschlüsselter Request:

```
{"header":0,
"sign":"VFhsU1lXUnBiMEo1VTNabGJnPT1NVFV5TWpNME16VXpOQT09",
"message":"S3lzclIybDJaVVJoZEdFckt5cz1NVFV5TWpNME16VXpOQT09"}
```

Unverschlüsselter Request:

```
{"header":0,
"sign":"MyRadioBySven",
"message":"+++GiveData+++"}
```

7. NACHRICHTENVERSCHLÜSSLUNG

7.1. ENCODE/DECODE

Jeder String der übertragen wird, wird vorher mittels **BASE64** encoded und decoded. („sync.encode“. „sync.decode“)

- Vorteil: Sicher und Stabil!

Für die BASE64 Verschlüsselung wurde ein „Salt-Key“ ergänzt, welcher die Verschlüsselung sicherer macht. Nur im besitzt des Keys können die Daten zurück gerechnet werden.

```
message = btoa(message + salt);          //z.B.1522343534
```

7.2. VERWENDUNG

Verwendet wird die Verschlüsslung an folgenden Punkten:

- Bei dem Empfang einer Korrekten Nachricht um den Sign-Tag im Header zu Prüfen.
- Bei richtigem Header zum Entschlüsseln der Nachrichten
- Zum Verschlüsseln vor dem Senden

Hierbei ist zu beachten, dass nicht der komplette String, der übertragen wird, auch verschlüsselt wird. Es wird nur jeweils unter dem JSON-String fallende „message“ und „sign“ verschlüsselt.

Beispiel:

```
{"header":0,
 "sign":"VFhsU1lXUnBiMEo1VTNabGJnPT1NVFV5TWpNME16VXpOQT09",
 "message":"S3lzclIybDJaVVJoZEdFckt5cz1NVFV5TWpNME16VXpOQT09"}
```

8. SPEICHERKONZEPTE

8.1. LOCAL-STORAGE

Für das persistente Speichern von Daten wird von der WebApi Storage, localStorage verwendet.
Dabei werden einmal die Favoriten unter „favorites“ und die Login-Daten unter „login“ gespeichert. Bei einer Registrierung wird, wie oben bereits beschrieben, der localStorage erweitert.Dies geschieht ebenfalls bei dem hinzufügen von Favoriten. Bei der Synchronisation werden Daten gelesen und abgeglichen.

Beispiel für ein User-Array mit JSON-Strings im localStorage unter dem Key: „login“:

```
[{"id":"4217441533",
   "name":"test",
   "password":3556498,
   "theme":0,
   "stamp":1497394850389},
{"id":"3345566657",
  "name":"moin",
  "password":3357255,
  "theme":0,
  "stamp":1497460162616}]
```

Beispiel für ein Favoriten-Array mit JSON-Strings im localStorage unter dem Key: „favorites“:

```
[{"id":"4217441533",
  "stamp":1497459488594,
  "list": [{"id":3791,
            "name":"Welle1Linz918",
            "country":"AT",
            "image":{"url":…,"thumb":{"url":…}},
            "categories":[{"title":"Top40"}],
            "facebook":null,
            "twitter":null,
            "website":"http://www.welle1.at/",
            "stream":[{"stream":"http://live.welle1.at:7128"}]},
        {"id":21458,
            "name":"AlexHits-90's",
            "country":"EG",
            "image":{"url":null,"thumb":{"url":null}},
            "categories":[{"title":"90s"}],
            "facebook":null,
            "twitter":null,
            "website":"",
            "stream":[{"stream":"http://s2.voscast.com:8698"}]
        }]
}
```

8.1. LOCAL-SESSION

Für das Speichern von none-persistenten Daten wird localSession verwendet. Hier werden der aktuelle Login-User unter „login“ und der aktuelle Stream unter „stream“ für eine Session gespeichert. Dies ermöglicht das eingeloggt bleiben, wenn die Seite neu geladen wird und das weiterspielen eines Stream für diesen Fall.

Beispiel für ein Stream und Login (JSON-Strings) im localSession unter dem Key: „stream“, „login":

```
{"stream":null,
 "obj":null}
```

```
{"name":"test",
 "password":3556498}
```

9. SYNCHRONISATION

Im Teil *Websocket* wurde bereits auf die Synchronisation eingegangen.

Für die Synchronisation werden die Daten nach der Übertragung überprüft. Alle Daten im localStorage haben einen Zeitstempel, der zur Überprüfung dient.

Benutzer:

Ist ein Benutzer nicht vorhanden, dann wird dieser ergänzt.
Ist der Zeitstempel aktueller, wird nur dieser Benutzer im Storage ersetzt. Dieser Fall kann auftreten, wenn der Benutzer zum Beispiel sein Theme ändert. Dadurch hat der Nutzer überall sein ausgewähltes Theme.

Favoriten:

Ist der Zeitstempel für die Favoriten zu einem Benutzer, der auch in Storage vorhanden ist, kleiner, dann werden diese Favoriten ersetzt.

Stream:

Der aktuelle Stream wird immer ersetzt.

10. FAZIT / BESONDERHEITEN

10.1. PROBLEME

Es sind einige Probleme bei dem Programmieren und Scripten aufgetaucht. Dies ist bei einem solchen Umfang sehr wahrscheinlich. Die meisten Fehler/Probleme sind nur sehr klein gewesen und konnten oft durch das Internet schnell gelöst werden.

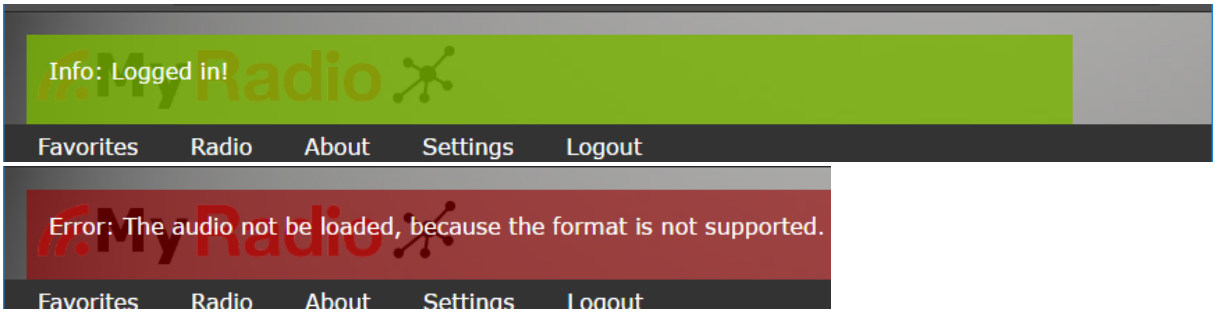
Ein größeres Problem war die Aufteilung der Datenpakete für die Synchronisation. Allerdings konnte dies mit viel Zeit/Geduld beim Ausprobieren gelöst werden.

Durch eine gute Dokumentation für die API zu Dirble und der Verbindung mit Borsti sind hier keine Fehler aufgetreten.

10.2. BESODNERHEITEN

- **Statusmeldungen / Fehlermeldungen:**

Bei erfolgreichen Aktionen werden hier entsprechende Meldungen ausgegeben. Diese werden kurzzeitig in einem Dialog eingeblendet. (Mittels „myRadio.checkOnlineState“ und myRadio.checkAudioError“)



- **Onlinecheck:**

Trotz fehlender Möglichkeit einen Server an zu pingen (Cross-Origin), ist es durch einen Button in der Filter-Leiste möglich, die Verfügbarkeit der aktuell angezeigten Liste zu prüfen. (TryAndError)

- **Sozialmedia:**

Bleibe auf dem laufenden durch die Sozialmedia-Links aus dem Streams, die mit angezeigt werden.

- **Stay in:**

Auch wenn du die Seite verlässt oder neu Lädst, wird durch eine Session dein Login kurzzeitig zurückgehalten und du bleibst drin! Genauso bekommst du deinen letzten Stream wieder, wenn dieser aktiv war.

- **Fancy Themes:**

Gutes Design der Seite (Bemerkungen aus Tests mit dritten) und angenehm ansehbare Themes (Beer)

- **Responsive:**

Die ganze Webapplication ist responsive aufgebaut. Dadurch ist sie für alle Geräte, sowie Smartphones aufrufbar.Dieses wurde mit CSS gelöst.

10.3. NICHT ERFÜLLT

- **Favoritenlisten (Eingeschränkt):**

Nur beschränkt erfüllt. Durch Missverständnis nur ein Favoritenlisten eingebaut. Es können keine mehrere Listen erstellt werden. Das heißt: Es kann mit einer, aber nur einer gearbeitet werden.

11. ABSCHLUSS

11.1. NAMESPACE

Für das die JS-Dateien wurden der Namespace bis auf zwei Funktionen eingehalten. Die einzigen Funktionen die Public sind:

- $(ID)
- tagName(NAME)

Diese wurden Public gemacht, da sie überall mehrfach gebrauch finden und keine sensiblen Daten verarbeiten.

11.2. RESPONSIVE

Die Seite ist durch ihr Responsive-Layout auch für mobile Geräte geeignet. Durch einen in CSS-gelösten Menü-Button lässt sich dort die ausgeblendete Navigation einfach einblenden.

11.3. SOFTWARE-REFERENZEN

Bei der Entwicklung wurden die IDEs *Brackets* und *WebStorm* verwendet. Als Browser kam nur *Google Chrome* zum Einsatz. Für die Bildbearbeitung wurde *GIMP2* genutzt. Für die Erstellung der Dokumentation wurde *Microsoft Word* genutzt.

11.4 QUELLENVERWEIS

Alle verwendeten Bilder und Informationen stammen von mir oder stehen unter einer Creative Commons Zero Lizenz (CC0) und sind somit frei für dritte.

Quellen von Favicons-Images:

www.facebook.com

www.twitter.com