JSP-Audioplayer – Gesamtdokumentation

Inhaltsverzeichnis

1. Kurze Gesamtbeschreibung

2. Audio-Player und Playlist (detailliert)

3. Flexibles Layout-Design (detailliert)

---

1. Kurze Gesamtbeschreibung

Erklärung des JSP-Audioplayers

Erstellt am: 06.05.2025

1. JSP Setup und Dateninitialisierung

Zu Beginn werden alle nötigen Java-Klassen importiert und die Parameter aus der URL gelesen. Die Hilfsklasse

```
Poster
```

wird verwendet, um Informationen über die verfügbaren Jahre und Runden zu erhalten.

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page import=... %>
...
Poster myPoster = new Poster(app);
String[] rounds = myPoster.GetAllRounds(year);
...
```

Anschließend werden Informationen für Navigation (z. B. Vorjahr, nächste Runde etc.) berechnet und als Session-Attribute gespeichert, sodass sie später auf der Seite oder in anderen JSPs wiederverwendet werden können.

2. HTML-Struktur und Seitenaufbau

Im

```
<head>
```

-Bereich werden die Stylesheets geladen und CSS-Regeln definiert. Diese strukturieren die Seite responsiv und sorgen für ein einheitliches Aussehen.

Das

```
<body>
```

enthält ein

```
<div class="container">
```

mit einem flexiblen Layout für Header, Content-Bereich und Footer. Das Layout passt sich der Höhe des Browserfensters an.

3. Header und Navigationsleiste

Im oberen Bereich wird ein Header mit Links zu Vor- und Folgejahr angezeigt. Falls Runden aktiv sind, gibt es darunter eine zweite Leiste mit Runden-Navigation.

```
<header class="header">
  <a href="audioplayer.jsp?year=...">« Vorjahr</a>
  ...
</header>
```

4. Audioverzeichnis und MP3-Verarbeitung

Abhängig vom Parameter

`round`

wird das passende MP3-Verzeichnis ermittelt. Dann werden alle MP3-Dateien geladen und mit

`mp3agic`

ausgelesen (z. B. Titel, Artist, Cover).

Falls ein Coverbild vorhanden ist, wird es als Base64 eingebettet und als Hintergrund verwendet. Es wird überprüft, ob es sich um ein Compilation-Album mit mehreren Künstlern handelt.

5. Audio-Player und Playlist-Tabelle

Der Player zeigt ein zentrales Steuerelement (Play, Nächstes, Zufall, Download) und eine Playlist-Tabelle mit allen Tracks. Die Tabelle enthält Coverbild, Titel und ggf. den Künstler.

6. Fehlerbehandlung

Falls keine MP3-Dateien gefunden werden, wird eine Zufallsgrafik angezeigt und eine Fehlermeldung ausgegeben.

7. JavaScript: Player-Steuerung

Das Skript im

`<script>`

-Block am Ende der Seite sorgt für:

- Playlist-Steuerung (Play, Zufall, Nächstes Lied, Herunterladen)
- Anzeige des aktuellen Songs (Titel, Künstler, Coverbild)
- Funktion zum Ändern des Seitenhintergrunds (aus dem Album-Cover)
- Zufällige Hintergrundfarben für Tabellenzellen

Besonderheiten: Base64-Bilder werden per

`Canvas`

transparent gemacht, was für den Hintergrundeffekt genutzt wird.

8. Footer und Seitenende

Am unteren Seitenrand wird erneut eine Navigationsleiste angezeigt, inklusive Link zurück zur „Interpreten“-Seite mit Buchstabenfilter.

Fazit

Die Seite ist modular aufgebaut und erlaubt die Navigation durch Jahre und Runden. Audio-Dateien werden elegant mit Metadaten präsentiert. Die Nutzung von Java, JSP und JavaScript ergibt ein dynamisches und anpassbares Medientool.

2. Audio-Player und Playlist (detailliert)

Detaillierte Beschreibung: Audio-Player und Playlist-Tabelle

Teil der JSP-Datei: Audioplayer mit MP3-Metadaten und Playlist-Darstellung

1. Grundstruktur des Players

Der Player ist innerhalb eines

`<main>`

-Elements eingebettet, das ein Hintergrundbild verwendet (aus dem ersten MP3-Cover entnommen). Er enthält drei Abschnitte:

- **Anzeige des aktuell gewählten Songs** (Titel, Künstler, Cover)
- **Steuerelemente**: Play, Nächstes Lied, Zufällige Wiedergabe, Download
- **Playlist-Tabelle**: Alle MP3s aus dem ausgewählten Verzeichnis

2. Playlist-Erzeugung aus MP3-Dateien

Die MP3-Dateien werden serverseitig mit

`mp3agic`

geladen. Pro Datei werden folgende Metadaten extrahiert:

- `getTitle()` : Liedtitel
- `getArtist()` : Interpret
- `getAlbumImage()` : Coverbild als Byte-Array

Falls kein Titel/Interpret vorhanden ist, werden Platzhalter wie "Unbekannter Künstler" angezeigt.

3. HTML-Aufbau der Playlist-Tabelle

Jede MP3-Datei erzeugt eine Tabellenzeile mit zwei Zellen:

- **Linke Spalte:** Entweder ein Mini-Coverbild (wenn vorhanden), oder einfach "1-10" Zählung bei einem Album.
- **Rechte Spalte:** Der Titel (als <h3>) und optional der Interpret bei Compilation-Alben.

```
<tr class="playlist-item" data-url=".../song.mp3">
  <td><img src="..."></td>
  <td><h3>Titel<br><span>Titel - Artist</span></h3></td>
</tr>
```

Das

`data-url`

-Attribut wird später im JavaScript verwendet, um den Pfad zum Song bereitzustellen.

4. JavaScript: Interaktive Steuerung

Die Buttons und Playlist-Einträge sind mit

`EventListener`

verknüpft:

- `#playAll`
  : Spielt alle Songs nacheinander ab.
- `#playNext`
  ,
  `#playPrev`
  : Navigation in der Liste.
- `#playRandom`
  : Zufällige Reihenfolge (mit Array-Shuffle).
- `#downloadFile`
  : Lädt den aktuell gewählten Song herunter.
- `.playlist-item`
  : Klick auf einen Song spielt diesen sofort ab.

Beim Abspielen wird das Album-Cover (falls vorhanden) als Hintergrundbild aktualisiert.

5. Zusätzliche visuelle Effekte

Das Album-Cover wird als Base64-encoded Bild auf einem Canvas gezeichnet, um die Transparenz anzupassen (z. B.

`opacity=0.3`

). Dadurch entsteht ein weicher Hintergrundeffekt.

Die Tabellenzellen erhalten außerdem eine zufällige Hintergrundfarbe, um die Seite dynamischer erscheinen zu lassen.

Fazit

Der Player ist komplett in JSP und JavaScript realisiert. Er nutzt serverseitige MP3-Verarbeitung und clientseitige Wiedergabe. Die Kombination aus MP3-Metadaten, dynamischer Playlist und visuellem Design ergibt eine benutzerfreundliche Audio-Anwendung für Webbrowser.

3. Flexibles Layout-Design (detailliert)

Flexibles Layout-Design des JSP-Audioplayers

1. Grundprinzip: Flexbox-Layout

Die Seite nutzt das CSS-Flexbox-Modell, um Inhalte vertikal zu organisieren. Das gesamte Layout ist innerhalb eines Containers organisiert:

```
.container (display: flex, flex-direction: column, height: 100dvh)
├── .header      (10%)
├── .header2     (optional, 5%)
├── .content / .content2 (75–80%)
└── .footer      (10%)
```

Dadurch bleibt die Seite stets vollständig im Viewport sichtbar und ist auf allen Bildschirmgrößen nutzbar.

2. Wichtige CSS-Regeln

1. `height: 100dvh;`
   – verwendet 100% der verfügbaren Fensterhöhe, auch auf mobilen Geräten.
2. `display: flex;`
   und

`flex-direction: column;`

– sorgt für eine vertikale Stapelung der Container.

3. `.content`

   nutzt

   `justify-content: center;`

   und

   `align-items: center;`

   für zentrierte Inhalte.
4. Responsives Schriftgrößen-Design durch:

   `max(1.7dvh, 9pt)`

   oder

   `min(4vh, 4vw)`

   .

3. Responsive Typografie und Tabellen

Die Schriftgrößen, Abstände und Tabellengrößen passen sich an die Bildschirmhöhe und -breite an. Beispiel:

```
td {
    font-size: max(1.7dvh, 9pt);
    background-color: rgb(238, 97, 97);
}
```

Das erlaubt sowohl auf kleinen Smartphones als auch auf großen Desktops eine gute Lesbarkeit.

4. Sticky Header für Tabellen

Mit folgenden Regeln wird der Tabellenkopf beim Scrollen fixiert, um die Orientierung zu erhalten:

```
.fixed-header thead {
    position: sticky;
    top: 0;
    z-index: 1;
}
```

5. Übersichtlicher Codeausschnitt mit Nummerierung

```
1. <div class="container" id="main">
2.   <header class="header">...</header>
3.   <main class="content">...</main>
4.   <footer class="footer">...</footer>
5. </div>
```

Fazit

Durch das durchdachte Flexbox-Design, dynamische Schriftgrößen und strukturierte CSS-Klassen wirkt die Seite modern, klar und funktioniert auf unterschiedlichsten Geräten einwandfrei.