

Mein MMP1 VERSE wurde mithilfe von vielen unterschiedlichen Technologien zum Leben erweckt.

HTML CSS JAVASCRIPT

Die Interaktivität und Benutzeroberfläche wurden mithilfe von HTML, CSS und JavaScript umgesetzt. Wobei für CSS unter anderem das Framework Tailwind CSS verwendet wurde, welches durch das Benennen der Klassen von Elementen den passenden CSS File generiert.

```
<div id="modal-background" class="fixed inset-0 bg-black bg-opacity-50 hidden"></div>
<div id="modal" class="fixed inset-0 flex items-center justify-center z-50 hidden">
<div class="bg-yellow-200 border-black border-2 rounded-lg p-8 max-w-xl">
<h2 id="modalh2" class="text-2xl font-bold mb-4"></h2>
<p id="modalmessage" class="mb-4">The correct answer was: <span id="modal-answer" class="font-bold"></span></p>
<button id="modal-close" class="bg-gray-700 text-white px-4 py-2 rounded-md">Close</button>
<button id="copy-text" class="bg-gray-700 text-white px-4 py-2 rounded-md mt-4">Copy Tries to Clipboard</button>
<button id="play-next-round" class="bg-gray-700 text-white px-4 py-2 rounded-md mt-4">Play Next Round</button>
</div>
</div>
```

Allerdings wurde auch reguläres CSS verwendet, um unter anderem Animationen und Responsivität zu regeln.

PHP

Um die serverseitige Logik zu implementieren, wurde PHP verwendet. Insbesondere um Kommunikation mit der Datenbank aufzubauen und Sessions zu verwalten.

POSTGRESQL PDO

Als Datenbank wurde Postgresql verwendet. Verwendet werden 4 Tabellen:

Songs (SongID, Title, Release-Date, Lyrics, ArtistFK, AlbumFK),
Artists(ArtistID,Name),
Album(Name,ArtistFK),
SongOfTheDay(ID)

```
$songId = $_SESSION['songId'];
$query = "SELECT songs.title, songs.lyrics, songs.release_date, albums.album_name, artists.artist_name
FROM songs
JOIN albums ON songs.album_id = albums.album_id
JOIN artists ON songs.artist_id = artists.artist_id
WHERE songs.song_id = :songId";

$statement = $dbh->prepare($query);
$statement->bindParam(':songId', $songId);
$statement->execute();
```

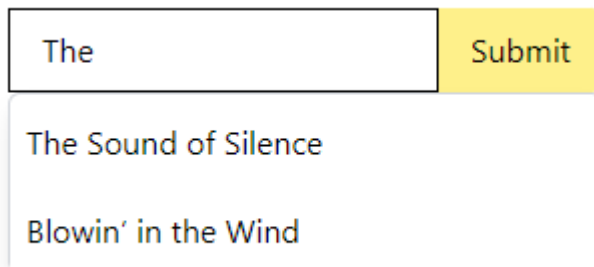
Die Lyrics werden in der Datenbank als Plain-Test gespeichert, und von PHP mit der Explode-Funktion automatisch in die einzelnen Zeilen aufgeteilt und in ein Array gespeichert.

JQUERY AJAX

Das Spiel umfasst Funktionen wie das Anzeigen der aktuellen Zeile, das Überprüfen der Benutzereingabe, das Zählen der Versuche und das Bereitstellen von Hinweisen bei Bedarf. Die Kommunikation zwischen dem Frontend und dem Backend erfolgte über AJAX-Anfragen von der JQuery library für eine reibungslose und dynamische Benutzererfahrung.

Die Verwendung von Sessions und dem Local-Storage verhindern Probleme mit Reloads.

Die AJAX Live-Search ermöglicht es, die Einträge aus der Datenbank als Vorschlag in der Suche zu erhalten, was die Usability erheblich steigert.



The

Submit

The Sound of Silence

Blowin' in the Wind

API

Daten für die Datenbank können direkt aus der Genius.com API geladen werden. In dem Script wird der Name und der Artist des Liedes übertragen, und durch PHP werden die erhaltenen Werte direkt in die Datenbank eingetragen

```
<?php
include "functions.php";

$api_key = "at5abbfZ54KVzFicB7CwYDPYMRNEqInIbAav7QxgRiVvPxLTpavP_m6z-1QG07He"; // Genius.com API token
$songTitleSearch = "losing my religion";
$artistNameSearch = "rem";

$encoded_query = urlencode("$songTitleSearch $artistNameSearch");

$url = "https://api.genius.com/search?q=$encoded_query";
$curl = curl_init($url);
curl_setopt($curl, CURLOPT_HTTPHEADER, ["Authorization: Bearer $api_key"]);
curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
$response = curl_exec($curl);

if ($response === false) {
    echo "Failed to fetch data from the Genius.com API.";
} else {
    $data = json_decode($response, true);

    $songInfo = $data['response']['hits'][0]['result'];
    $songId = $songInfo['id'];
    $songTitle = $songInfo['title'];

    $url = "https://api.genius.com/songs/$songId";
    $curl = curl_init($url);
    curl_setopt($curl, CURLOPT_HTTPHEADER, ["Authorization: Bearer $api_key"]);
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
    $response = curl_exec($curl);
```

CRONTAB

Im Spiel wird auch ein „Tägliches Quiz“ angeboten welches sich automatisch jeden Tag um Mitternacht ändert. Verwendet dafür wird Crontab am Webserver. Um 00:00 jedes Tages wird das Script `updateTodazSong.php` automatisch ausgeführt, welches das Quiz des Tages setzt.

```
DEBUG CONSOLE  PROBLEMS  OUTPUT  PORTS  SQL CONSOLE  TERMINAL

0 * * * /usr/bin/php /home/store/fhs49049/webpace_mmt/MMP1/final/updateTodaySong.php
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow   command
"/tmp/crontab.VRPB8U/crontab" 23L, 977C
```

ORDNERSTRUKTUR

```
EXPLORER
...
▼ FINAL
  > .vscode
  > dist
  > doc
  > node_modules
  > src
  > textures
  📄 changesong.php
  📄 config.php
  JS dailyquiz.js
  📄 dailyquiz.php 2
  📄 dailysong.php
  📄 footer.php
  📄 functions.php
  📄 header.php 2
  JS headermodal.js
  📄 impressum.php
  📄 index.php
  📄 linesplitter.php
  📄 loadAPI.php
  📄 local.session.sql
  📄 mmp1.session.sql
  {} package-lock.json
  {} package.json
  JS randomquiz.js
  📄 randomquiz.php 2
  JS script.js
  📄 search.php
  JS tailwind.config.js
  📄 updateTodaySong.php
  📄 verify-answer.php
```