

Corso di Programmazione Web e Mobile

A.A. 2021/2022



LyricsGuesser.net

Rover Simone 933288

Bartolomei Daniel 922804

1. Introduzione

Ispirato a numerosi giochi televisivi e passatempi estivi, LyricsGuesser consente di giocare su browser al popolare “indovina la canzone” avvalendosi delle sempre più crescenti informazioni reperibili sul web, come, appunto, le lyrics.

LyricsGuesser mette alla prova l’utente fornendo testi di canzoni prelevate dalle playlist più popolari e classifiche globali o direttamente selezionate dall’account Spotify del giocatore.

1.1 Breve analisi dei requisiti

1.1.1 Destinatari

Capacità e possibilità tecniche

Il servizio è facilmente fruibile da qualsiasi pubblico, ma punta a soddisfare gli utenti che, avendo conoscenze più generali dell’industria musicale, possono trarre maggiore intrattenimento nell’indovinare i brani da loro seguiti.

Inoltre, funge da gioco di gruppo, nel tentativo di totalizzare il punteggio più alto possibile.

Il sito è ottimizzato per desktop, ma è comunque accessibile da mobile grazie ad accorgimenti implementati sull’interfaccia.

Linguaggio

La lingua adottata dal servizio è l’inglese, in modo che l’esperienza sia indipendente dal luogo di provenienza.

Motivazione

Il sito ha uno scopo intrattenitivo e educazionale tramite la repository pubblica in cui è possibile consultare il codice sorgente dell’intero progetto.

1.1.2 Modello di valore

Il valore dell'applicazione deriva, oltre che dall'ampia utenza che può interagire con la piattaforma, dalla possibilità di accedere alla pagina attraverso Spotify e alla personalizzazione dell'esperienza.

1.1.3 Flusso dei dati

Ottenere i contenuti

I dati e i testi delle canzoni sono reperiti sul web, tramite API fornite da Spotify e Musixmatch.

A seconda del tipo di accesso scelto, i dati del giocatore vengono prelevati da Spotify oppure forniti direttamente dall'utente.

Siccome i testi delle canzoni sono oggetto di copyright, per avere un servizio al 100% funzionale occorrerebbe acquistare una chiave per ottenere la copertura di copyright da parte di Musixmatch.

Nel nostro caso otteniamo il 30% del testo di una canzone senza alcun costo, ma con divieto d'uso commerciale e un numero ridotto di richieste effettuabili su base giornaliera.

Archiviare e organizzare i contenuti

Data la bassa quantità di dati da memorizzare per utente, abbiamo deciso di optare per un sistema di archiviazione in locale sul file system della macchina ospitante il sito.

I dati sensibili sono criptati utilizzando SHA256, per un futuro passaggio ad HTTPS.

Pubblicare i contenuti

La modalità di gioco che definisce che brani verranno presentati all'utente è scelta sempre da quest'ultimo.

Alcune modalità presenti hanno un catalogo di brani a disposizione che è variabile, seguendo i cambiamenti giornalieri che avvengono su Spotify.

Con la visualizzazione del testo di una canzone, segue obbligatoriamente la visualizzazione delle informazioni di copyright, che non dovrà mai essere celata.

Inoltre, includiamo il Pixel Tracking fornito da Musixmatch nella pagina di gioco come previsto dalla documentazione.

1.1.4 Aspetti tecnologici

Standard per i contenuti

Il formato JSON è utilizzato per l'archiviazione delle informazioni in locale.

La **view** è implementata dai file HTML e CSS, mentre il **controller** che si occupa di gestire gli input dell'utente è caratterizzato da Javascript con chiamate AJAX al server di backend, che a sua volta implementa il **model**.

In particolare, il **model** è implementato tramite un'applicazione NodeJS che gestisce i dati archiviati e le chiamate a API esterne a seconda degli input ricevuti dal **controller**

Tecnologie utilizzate

- **HTML5 e CSS:**

Utilizzati per implementare le interfacce utente e gli stili di pagina. Usiamo LocalStorage di HTML5 per memorizzare dati riguardanti l'utente loggato alla piattaforma.

- **Javascript e AJAX:**

AJAX è la componente fondamentale del codice lato client col quale otteniamo dati per popolare le pagine del sito.

In particolare, usato per salvare dati utente (es: registrazione al sito) e ottenere le lyrics dei brani.

- **APIs:**

Il servizio interagisce con le API di Spotify e Musixmatch.

- **NodeJS:**

Utilizziamo NodeJS per implementare il server di backend e fornire segretezza a dati sensibili o che non vogliamo siano visibili all'utente in fase di gioco (es: titolo della canzone da indovinare).

In particolare, abbiamo utilizzato Express e Axios: il primo per interfacciarsi con il sito di LyricsGuesser e il secondo per inoltrare le API calls.

- **Apache:**

Il server Apache viene usato per ospitare il lato client dell'applicazione. Una macchina Debian11 ospita sia il server Apache sia il server NodeJS.

- **Google Domains:**

È stato acquistato il dominio lyricsguesser.net tramite il servizio di Google Domains e configurato il DNS.

- **Github:**

Viene usato per rendere accessibile il codice sorgente del sito e del backend server.

1.1.5 Altro

FontAwesome

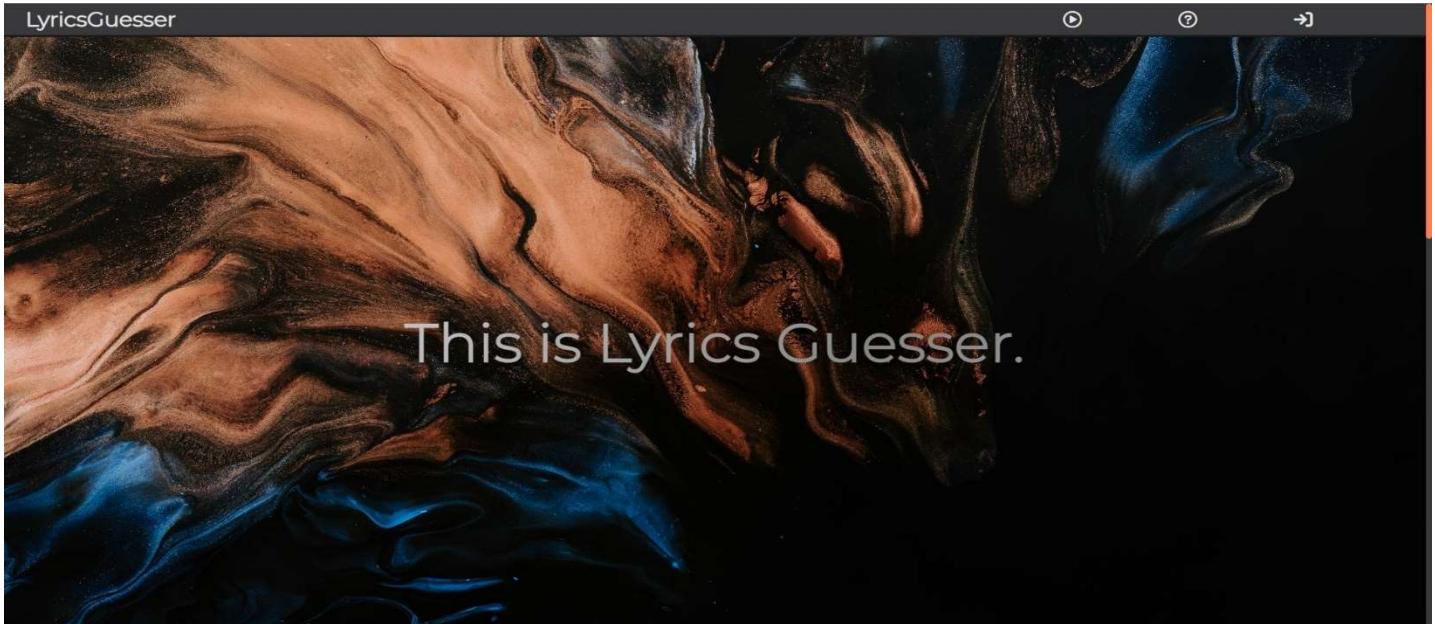
Abbiamo utilizzato alcune icone fornite dal sito fontawesome.com

TermsFeed

Abbiamo utilizzato il servizio di termsfeed.com per generare le policy riportate sul sito di LyricsGuesser.

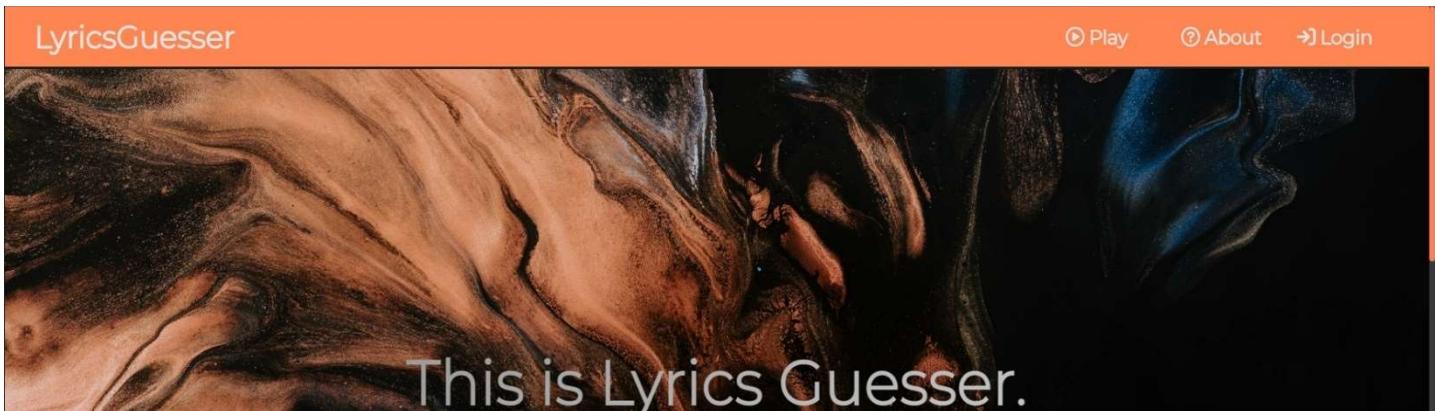
2. Interfacce

Pagina Principale



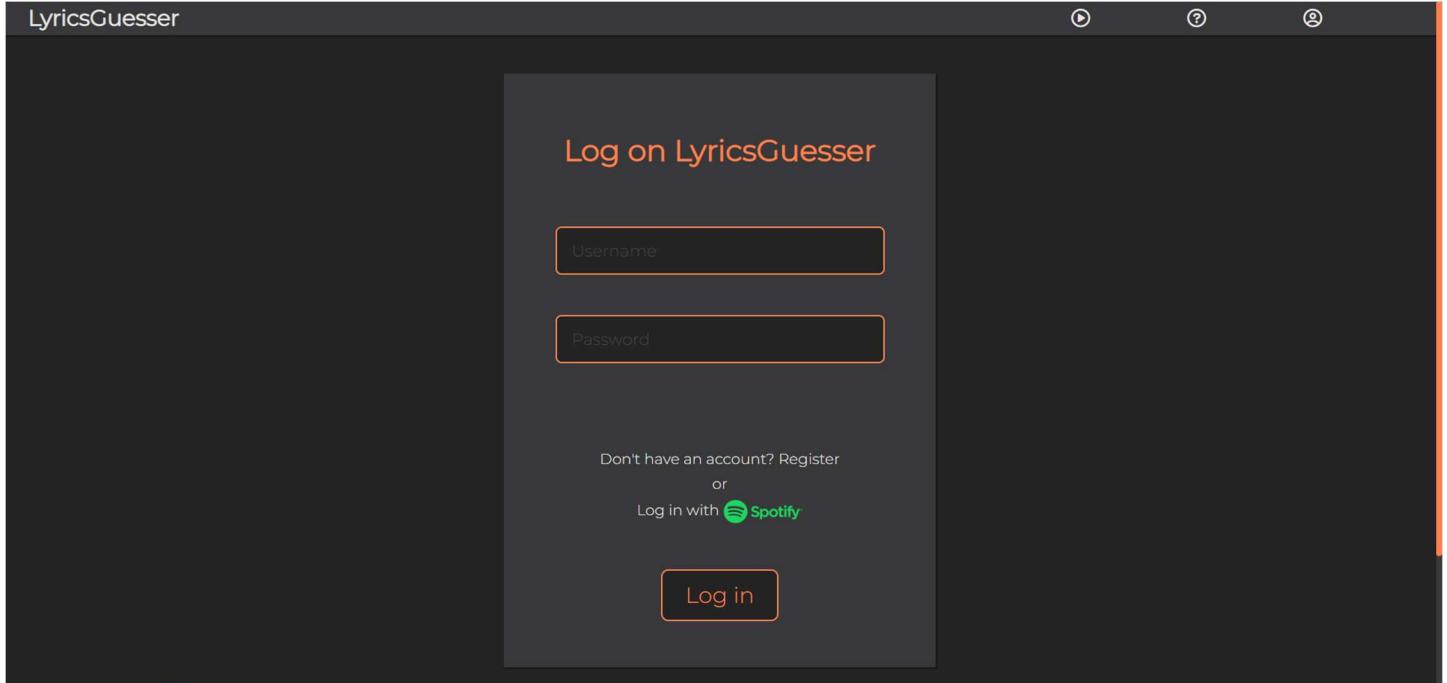
La pagina principale è strutturata con un effetto di parallasse a scorrimento e di gradiente sul testo che fornisce una breve introduzione della pagina all'utente.

Navbar



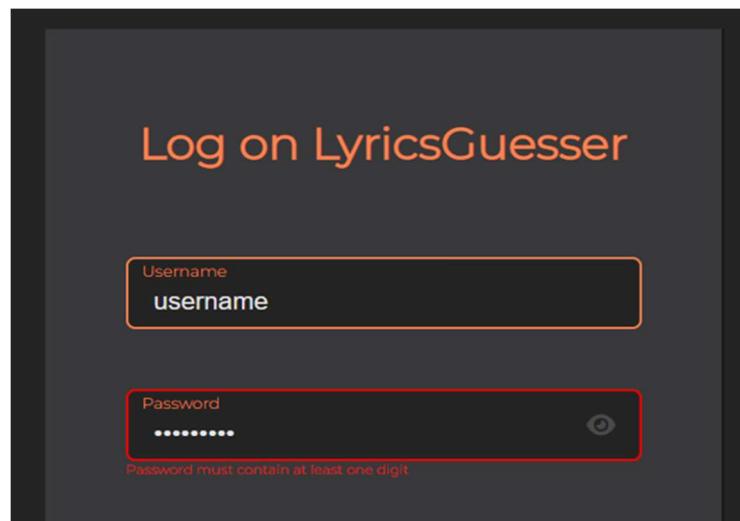
La navbar in testa alla pagina è presente in tutte le pagine del sito e ha un effetto di hover, ingrandendosi e dando accesso al menù di navigazione in alto a destra.

Pagina di Login



La pagina di login presenta collegamenti alla Pagina di Registrazione e all'accesso attraverso Spotify.

Inoltre, in caso di credenziali errate queste vengono evidenziate.



Le stesse caratteristiche valgono per la Pagina di Registrazione.

Pagina di selezione della Modalità senza Accesso e Footer

The screenshot shows the LYRICSGUESSER MODES section of the website. At the top, there is a white header bar with the text "LyricsGuesser" on the left and three small icons on the right. Below the header, a white rectangular box contains the text "LYRICSGUESSER MODES". Underneath this, a large orange rectangular box contains the text "REGISTER or LOGIN with a LyricsGuesser account to play!". At the bottom of the page, there is another white rectangular box containing the text "SPOTIFY MODES". Below this, a green rectangular box contains the text "Login with Spotify in order to get access to other gamemodes!".

REGISTER or LOGIN with a LyricsGuesser account to play!

SPOTIFY MODES

Login with Spotify in order to get access to other gamemodes!

Github Disclaimer Terms Privacy Contacts

LyricsGuesser © 2022

La pagina di selezione avrà questo aspetto quando non è stato effettuato l'accesso né tramite LyricsGuesser stesso né tramite Spotify.

I banner contengono link che rimandano alle pagine inerenti.

È visibile anche il footer del sito, che presenta collegamenti inerenti alle informazioni del progetto o di privacy.

Questo è presente in ogni pagina, esclusa la pagina di gioco.

Pagina di selezione della Modalità

The screenshot displays the 'LyricsGuesser' application interface. At the top, there is a dark header bar with the app's name and three small icons. Below the header, the main content area is divided into several sections:

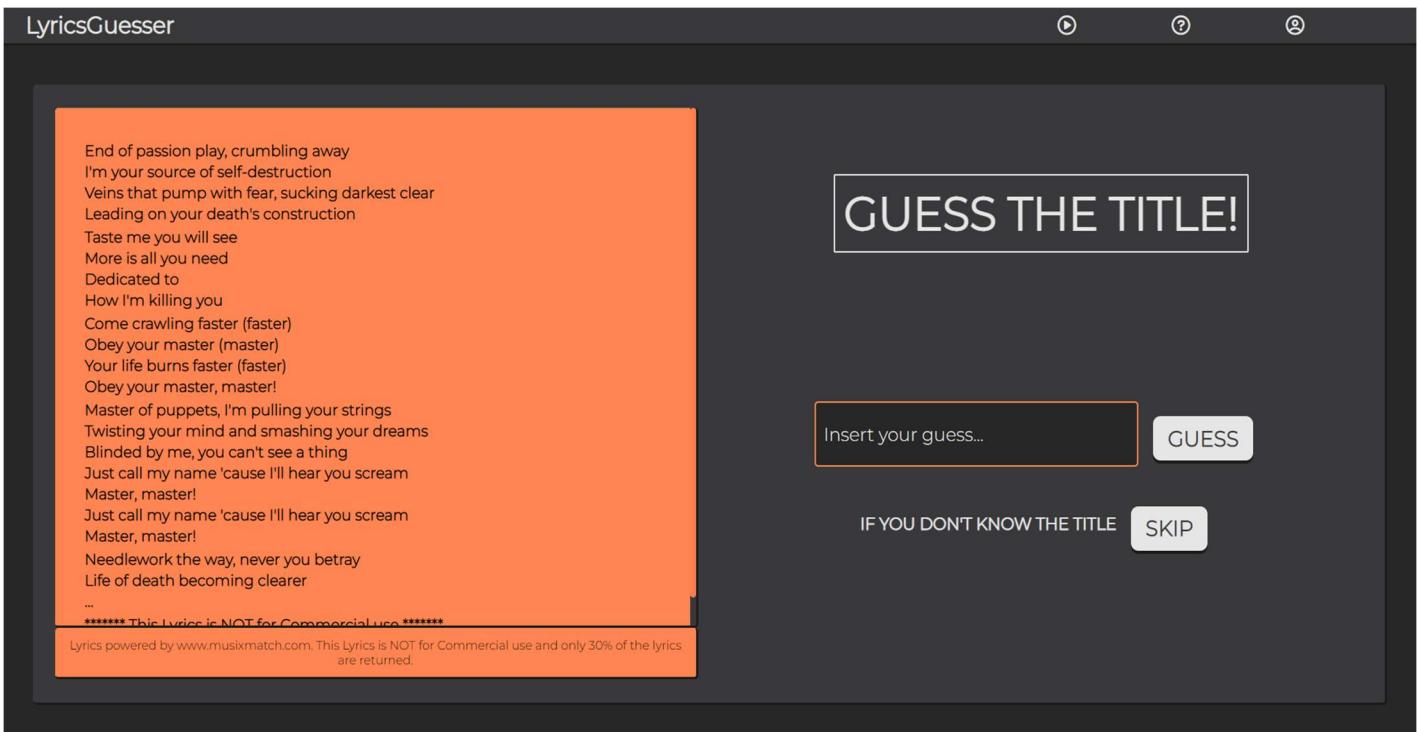
- GENERAL MODES**: This section contains three cards, each representing a different mode:
 - TOP 50 GLOBAL**: You'll be guessing songs from Spotify's TOP 50 GLOBAL playlist.
 - TODAY'S TOPs**: You'll be guessing songs from Spotify's TOP songs in the last 24 hours.
 - INTERNATIONAL HITS**: You'll be guessing songs from Spotify's INTERNATIONAL HITS playlist.
- MODES BY YEAR**: This section contains two cards:
 - 2010s SONGS**: You'll be guessing songs from Spotify's ALL-OUT-2010 playlist.
 - [A partially visible card]
- SPOTIFY MODES**: This section contains two cards, both highlighted with a green background:
 - FAVORITES**: You'll be guessing songs from your own Spotify FAVOURITES playlist.
 - PLAYLISTS**: You'll be guessing songs from any of your Spotify playlist, created or followed.

At the bottom of the screen, there is a footer navigation bar with links to [Github](#), [Disclaimer](#), [Terms](#), [Privacy](#), and [Contacts](#). A small copyright notice, "LyricsGuesser © 2022", is also present in the footer.

Questo è l'aspetto della Pagina di Selezione quando l'utente ha eseguito l'accesso e ha a disposizione entrambe le liste di modalità.

I banner hanno effetti stilistici che rendono la pagina più responsiva.

Pagina di Gioco

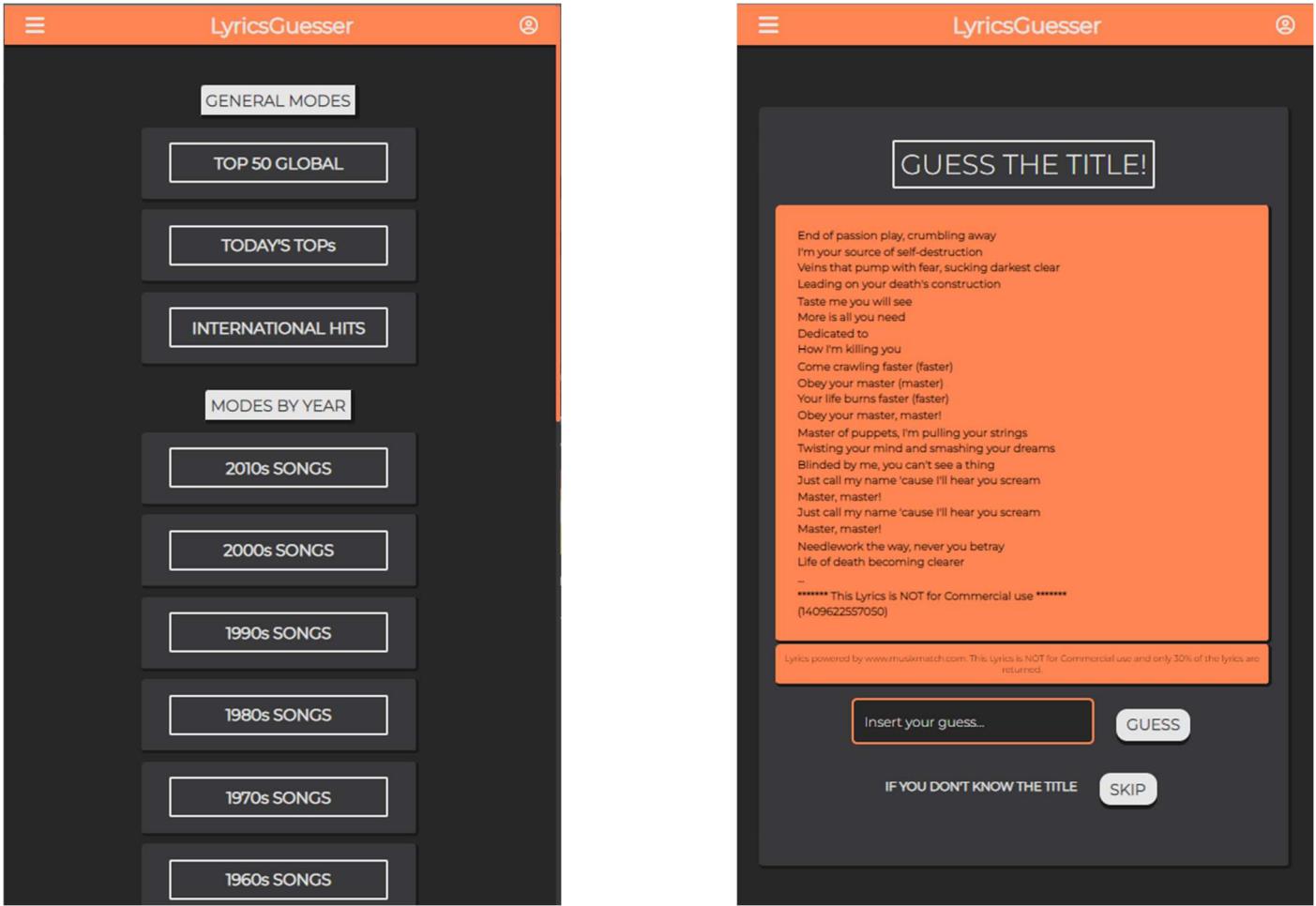


La schermata di gioco è suddivisa in due parti chiave, il contenitore che presenta all'utente le lyrics che dovrà indovinare e l'interfaccia con cui l'utente interagisce con il gioco.

Da notare che sotto il contenitore delle lyrics sarà sempre presente, per motivi legali, un disclaimer fornito da Musixmatch.

Nel momento in cui l'utente dà la propria risposta, verrà visualizzato un popup contenente punteggio ottenuto per l'input inserito, il titolo effettivo del brano e valore totale i punti ottenuti.

Schermate adattate per Mobile



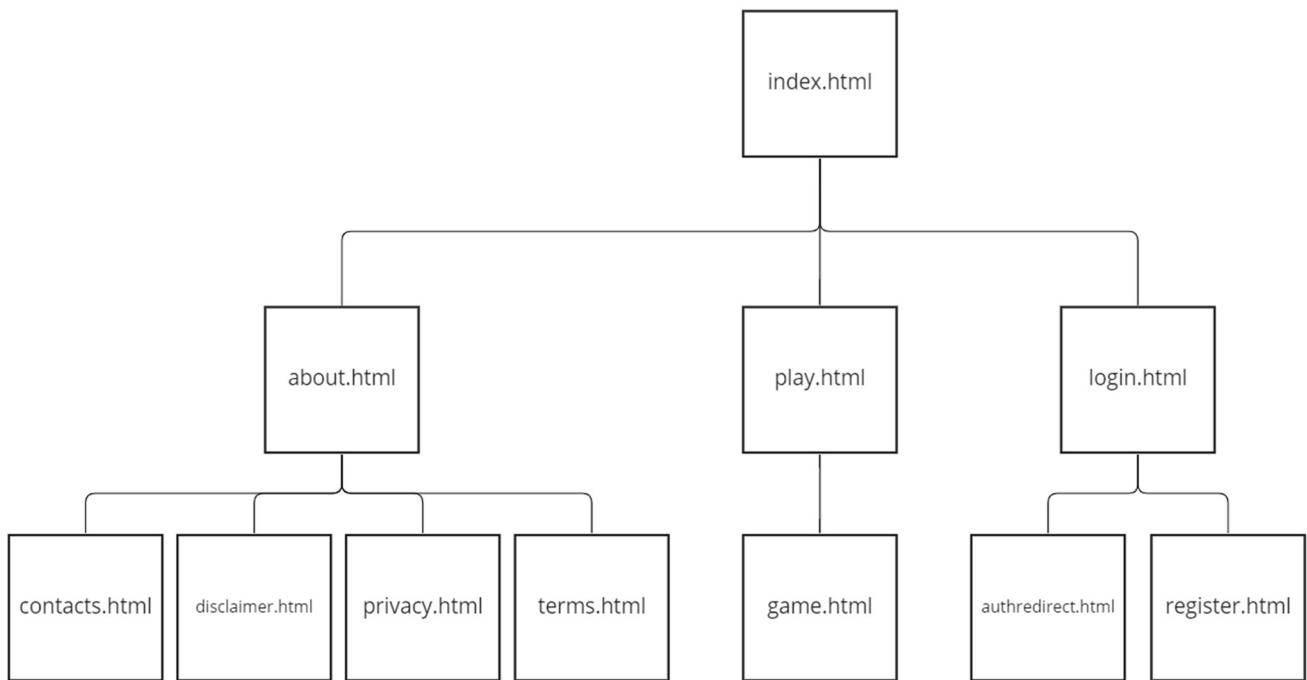
La differenza principale è che la navbar rimane di dimensione fissa e i menu di navigazione sono raccolti in un overlay, accessibile dall'apposito tasto in alto a sinistra.

Rimane visibile in alto a destra il tasto di Login o User nel caso abbia già effettuato l'accesso.

3. Architettura

3.1 Diagramma dell'ordine gerarchico delle risorse

Identificazioni delle pagine da rendere accessibili come risorse (URI) e loro organizzazione gerarchica.

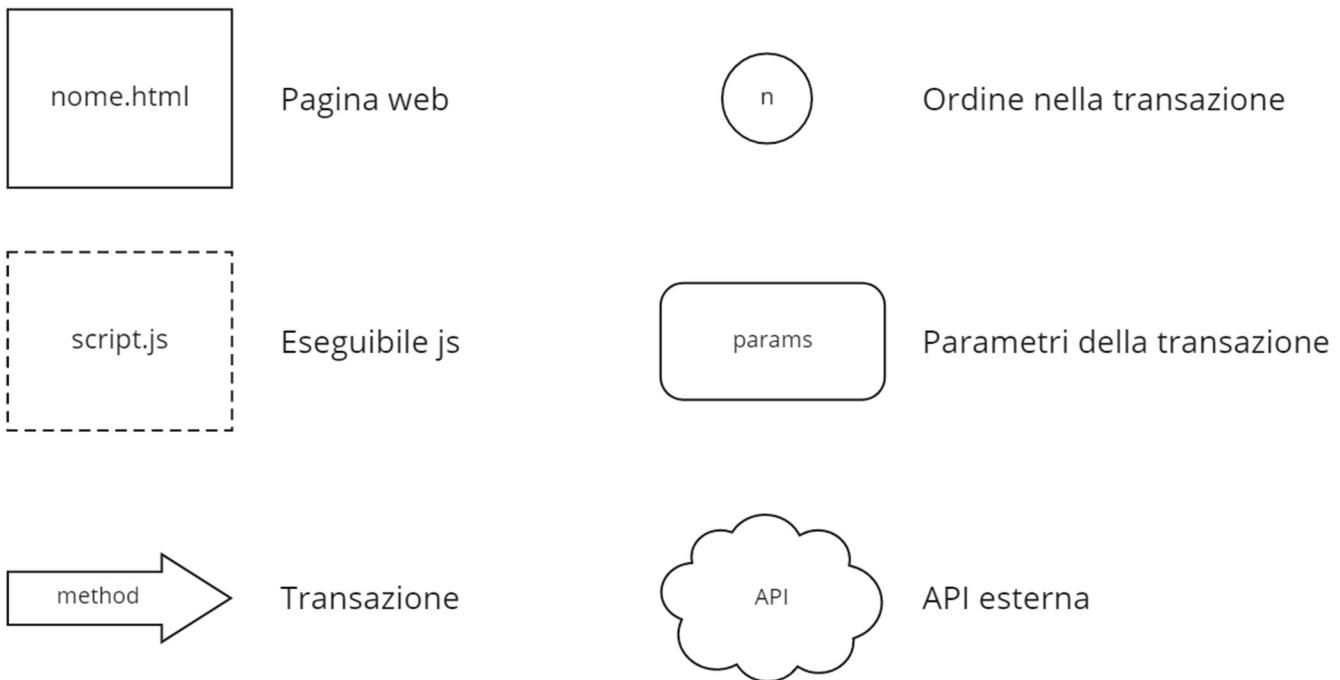


3.2 Descrizione delle risorse

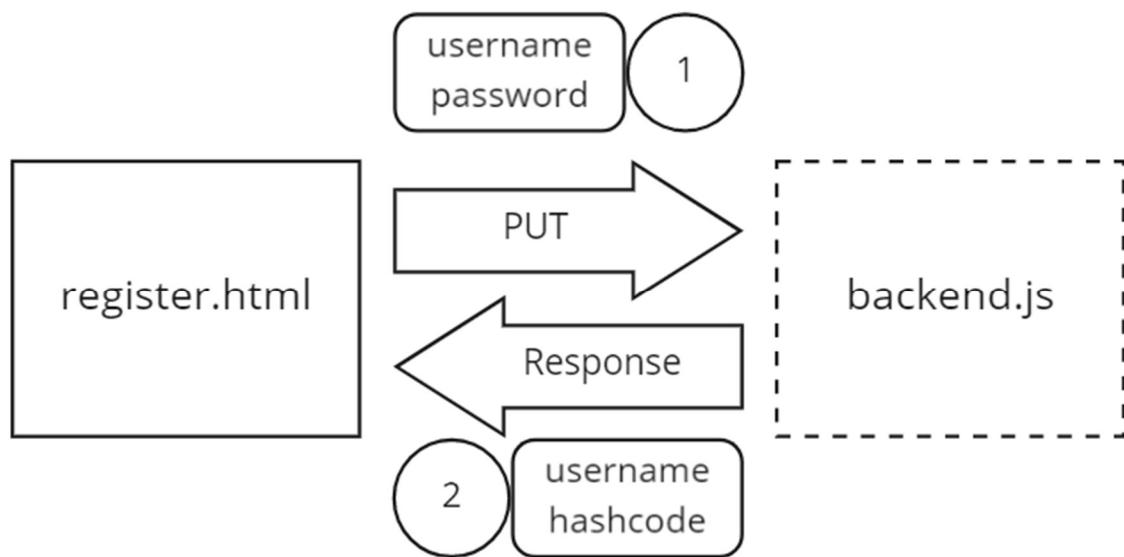
Di seguito elenchiamo le possibili transazioni tra frontend-backend e backend-API.

Ogni pagina HTML ha un corrispondente script con lo stesso nome che viene usato come controller.

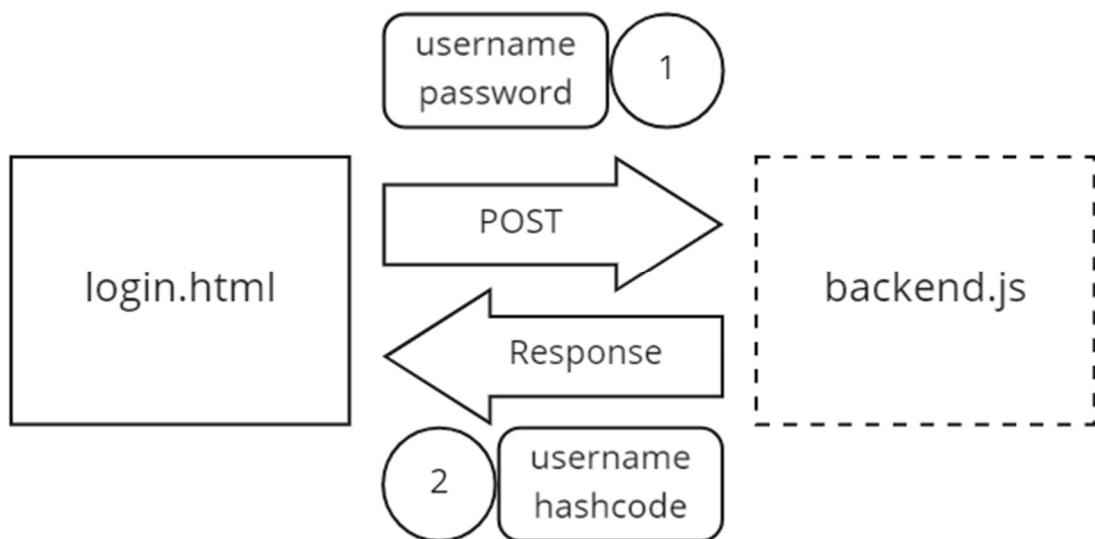
Legenda



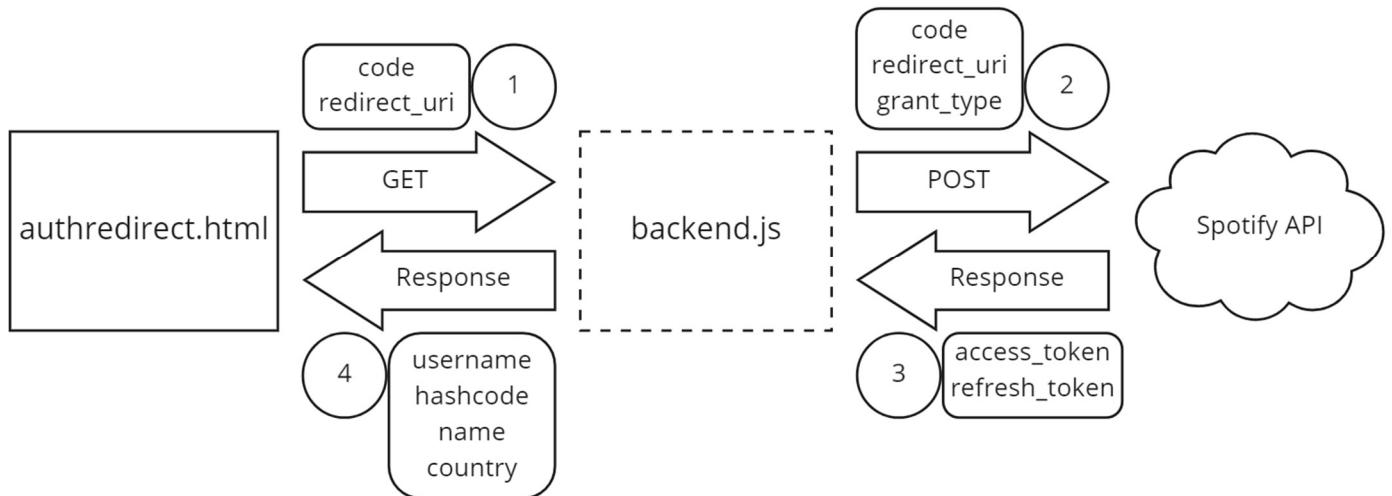
Registrazione di un utente su LyricsGuesser



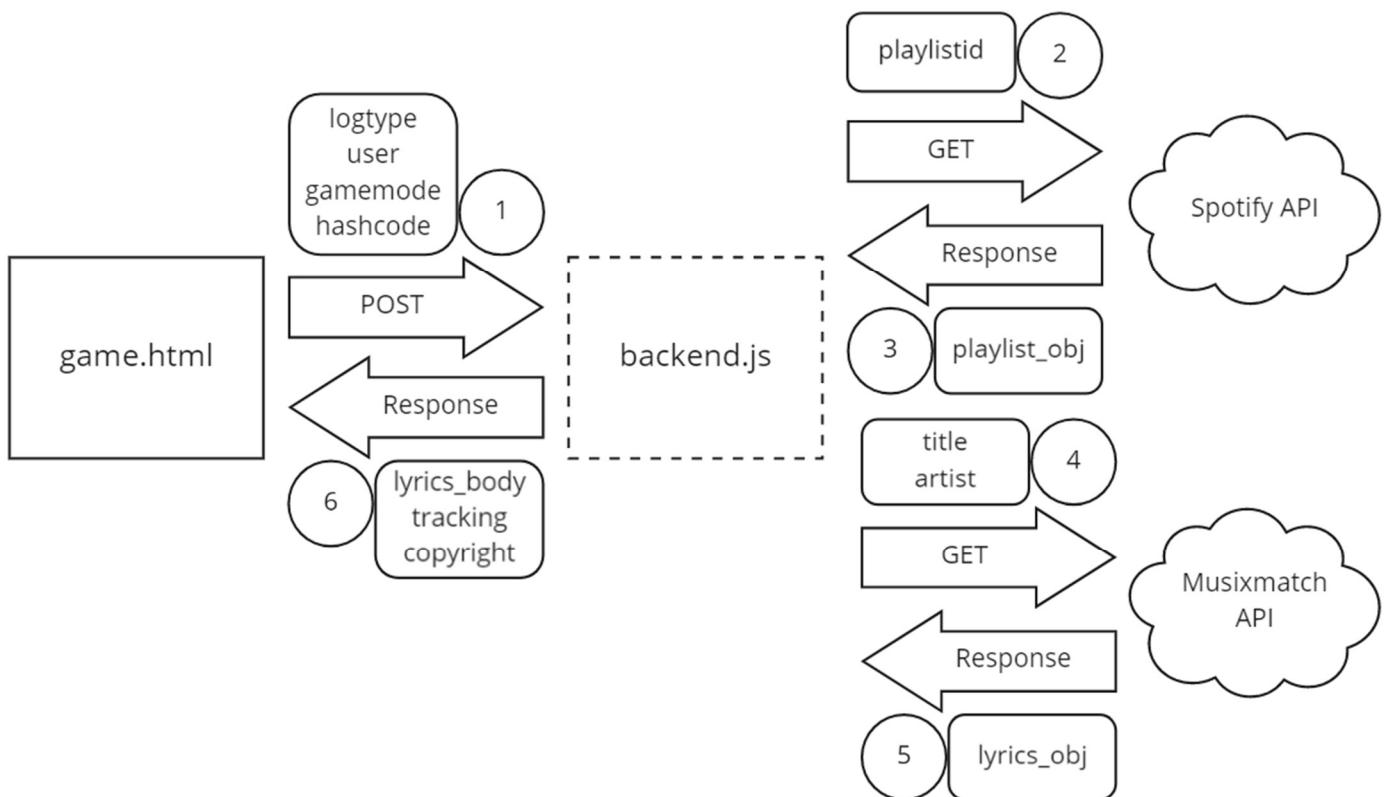
Login con LyricsGuesser



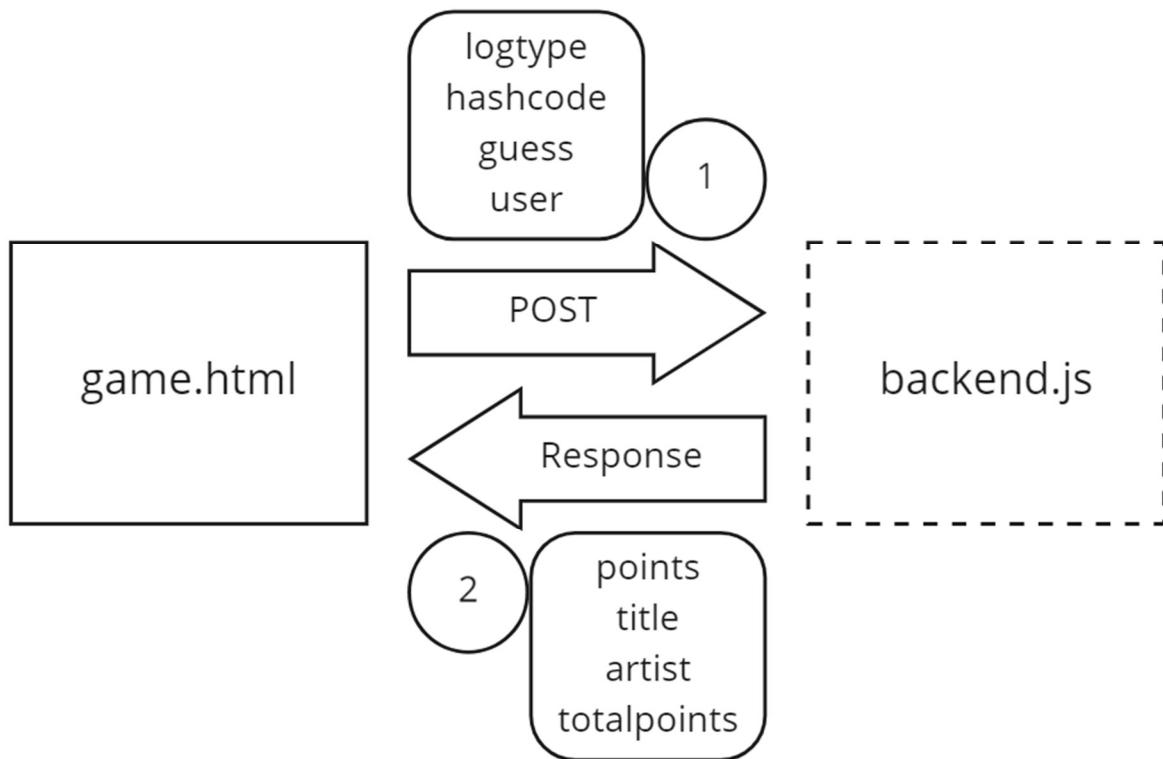
Login con Spotify



Richiesta del testo di una canzone



Submit risposta utente per una canzone



4. Codice

4.1 Frammenti di codice più significativi

4.1.1 HTML

Navbar

La navbar è predisposta per due layout: desktop e mobile.

Gli elementi custom-bars e nav-link-custom sono visibili solo su dispositivi mobili, quando la lista dei link viene oscurata.

```
<!-- Navbar -->
<nav id="myNav">
    <a class="custom-bars" onclick="OpenNav()"><i class="fa-solid fa-bars"></i></a>

    <a href="/" class="nav-title">LyricsGuesser</a>

    <ul class="nav-links">
        <li>
            <a href="/pages/play.html" class="nav-link">
                <i class="fa-regular fa-circle-play"></i>
                <span class="link-label">Play</span>
            </a>
        </li>

        <li>
            <a href="/pages/about.html" class="nav-link">
                <i class="fa-regular fa-circle-question"></i>
                <span class="link-label">About</span>
            </a>
        </li>

        <li>
            <a href="/pages/login.html" class="nav-link" id="trigger">
                <i class="fa-solid fa-arrow-right-to-bracket"></i>
                <span class="link-label">Login</span>
            </a>
        </li>
    </ul>

    <a href="/pages/login.html" class="nav-link-custom" id="trigger-mobile">
        <i class="fa-solid fa-arrow-right-to-bracket"></i>
    </a>
</nav>
```

Form di login

Il form di login, oltre alle textbox con le label utilizzate come placeholder, ci sono gli elementi di pagina per la comunicazione con l'utente.

Le text-tooltip fanno capire all'utente se il formato dello username o password è errato, mentre login-result comunica l'esito del login.

```
<div class="register-form">
  <div class="form-title">
    Log on LyricsGuesser
  </div>
  <div>
    <div class="register-textbox">
      <div class="textbox">
        <input type="text" name="username" id="username" class="expecting-input" autocomplete="off">
        <label for="username" id="username-label">Username</label>
      </div>
      <div class="textbox-tooltip" id="username-tooltip">
        Username not valid
      </div>
    </div>

    <div class="register-textbox">
      <div class="textbox">
        <input type="password" name="password" id="password" class="expecting-input">
        <label for="password" id="password-label">Password</label>
        <i class="fa-solid fa-eye" id="show-password" onclick="ShowPassword()"/>
      </div>
      <div class="textbox-tooltip" id="password-tooltip">
        Password not valid
      </div>
    </div>
  </div>
  <div class="register-result" id="login-result">
    Login successful <i class="fa-solid fa-check"></i>
  </div>
  <div class="login-container">
    <div class="login-text">
      Don't have an account? <a class="login-link" href="register.html">Register</a>
    </div>
    <div class="login-text middle-login-text">
      or
    </div>
    <div class="login-text">
      <a class="login-link" href="https://accounts.spotify.com/it/authorize?client_id=5e87bce380b34915a">Log in</a>
    </div>
  </div>
  <div class="button-container">
    <a class="register-button" onclick="Validate()">Log in</a>
  </div>
</div>
```

4.1.2 CSS

Effetto gradiente del logo nella homepage

```
/* Logo animation */
.animate-character
{
    background-image: linear-gradient(
        -225deg,
        #858585 0%,
        #cacaca 40%,
        #ffffff 60%,
        #cacaca 80%,
        #858585 100%
    );
    background-size: auto auto;
    background-clip: border-box;
    background-size: 200% auto;
    color: #ffffff;
    background-clip: text;
    -webkit-background-clip: text;
    -webkit-text-fill-color: transparent;
    animation: textclip 3s linear infinite;
}

@keyframes textclip {
    to {
        background-position: 200% center;
    }
}
```

4.1.3 API

Spotify API

Funzione 1: Fetch dell'access-token di LyricsGuesser da Spotify API

```
async function fetchAppAccessToken(){
  let encrypted = Buffer.from(process.env.SPOTIFY_CLIENT_ID + ':' + process.env.SPOTIFY_CLIENT_SECRET);
  let options = {
    url: SPOTIFY_TOKEN_ENDPOINT,
    method: "POST",
    data: "grant_type=client_credentials",
    headers: {
      "Authorization": "Basic " + encrypted.toString("base64"),
      "Content-Type": "application/x-www-form-urlencoded"
    }
  }
  try {
    const resp = await axios.request(options);
    return resp.data.access_token;
  }
  catch (error) {
    console.log(error);
    return null;
  }
}
```

Funzione 2: Richiesta dei brani di una playlist pubblica di Spotify

```
async function RequestPublicSpotifyPlaylistTracks(playlistid, limit = 1, offset = 0) {
  if (playlistid === null || playlistid === undefined) return null;
  if (offset < 0 || offset === null) offset = 0;
  if (limit < 1) limit = 1;
  if (limit > 50) limit = 50;
  let options = {
    url: SPOTIFY_API + SPOTIFY_PLAYLIST_ENDPOINT + '/' + encodeURI(playlistid) + "/tracks?limit=" + limit + "&offset=" + offset,
    method: "GET",
    headers: {
      "Authorization": "Bearer " + APP_ACCESS_TOKEN
    }
  }
  let resp;
  try {
    resp = await axios.request(options);
  }
  catch (error) {
    if (error.response.status == 401) {
      APP_ACCESS_TOKEN = await fetchAppAccessToken();
      resp = await RequestPublicSpotifyPlaylistTracks(playlistid);
    } else if (error.response.status == 429) {
      console.log("Spotify API limit reached");
      resp = error.response;
    } else {
      resp = error.response;
    }
  }
  return resp;
}
```

Funzione 3: Richiesta delle playlist di un utente

```
async function GetUserPlaylists(userid, limit = 1, offset = 0) {
    if (userid === null) return null;

    let userTokens = GetUserTokensFromId(userid);
    if (userTokens === null) return null;

    if (offset < 0 || offset === null) offset = 0;
    if (limit < 1) limit = 1;
    if (limit > 50) limit = 50;

    let options = {
        url: SPOTIFY_API + "/me/playlists?limit=" + limit + "&offset=" + offset,
        method: "GET",
        headers: {
            "Authorization": "Bearer " + userTokens.access_token
        }
    }
    let resp;
    try {
        resp = await axios.request(options);
    }
    catch (error) {
        if (error.response.status == 401) {
            console.log("Need to refresh usertoken")
            await RefreshUserToken(userTokens.refresh_token, userid);
            resp = await GetUserPlaylists(userid, limit, offset);
        } else if (error.response.status == 429) {
            resp = error.response;
            console.log("Spotify API limit reached");
        } else {
            resp = error.response;
        }
    }
    return resp;
}
```

Musixmatch API

Viene utilizzata per ottenere i dati di lyrics e copyright

Funzione 4: Richiesta dei dati di lyrics di un brano all'API di Musixmatch

```
async function RequestSongLyrics(songTitle, artist) {
  if (songTitle === null || artist === null) return null;
  let url_temp = MUSIXMATCH_ENDPOINT + MATCHER_ENDPOINT + "?apikey=" + process.env.MUSIXMATCH_API_KEY
  url_temp += "&q_track=" + encodeURI(songTitle) + "&q_artist=" + encodeURI(artist);
  let options = {
    url: url_temp,
    method: "GET"
  };
  let resp;
  try {
    resp = await axios.request(options);
  } catch (error) {
    if (error.response.status == 402) {
      console.log("Musixmatch API limit reached");
    }
    resp = error.response;
  }
  return resp;
}
```

4.1.4 Javascript

Come esempio dell'utilizzo di Javascript riportiamo le procedure che implementano il fetch di una lyrics e l'update del view.

Funzione 5: Richiesta di un testo da game.js al server di backend

```
function RequestLyrics(reqBody){  
    return new Promise(function (resolve, reject) {  
        let lyricsReq = new XMLHttpRequest();  
        lyricsReq.open("POST", "http://lyricsguesser.net:8000/getSongToGuess", true);  
        lyricsReq.timeout = 1500;  
        lyricsReq.setRequestHeader("Content-type", "application/json");  
        lyricsReq.onload = () => {  
            if (lyricsReq.status == 200) {  
                resolve(JSON.parse(lyricsReq.response));  
            } else {  
                reject({  
                    "status": lyricsReq.status,  
                    "statusText": lyricsReq.statusText  
                });  
            }  
        };  
        lyricsReq.onerror = () => {  
            reject({  
                "status": lyricsReq.status,  
                "statusText": lyricsReq.statusText  
            });  
        }  
        lyricsReq.ontimeout = () => {  
            reject({  
                "status": 504,  
                "statusText": "Gateway timeout"  
            });  
        }  
        lyricsReq.send(JSON.stringify(reqBody));  
    });  
}
```

Funzione 6: Collezione dei dati per la richiesta e gestione della risposta

```
async function GetLyricsData(){
    let params = new URLSearchParams(window.location.search);

    let logType = localStorage.getItem("logtype");
    let hashCode = localStorage.getItem("hashcode");
    let gameMode = params.get("gamemode");
    let user = localStorage.getItem("user");

    if (logType === undefined || hashCode === undefined || gameMode === undefined || gameMode === null || user === undefined){
        return {"status": 400, "statusText": "Bad request"};
    }
    let reqBody = {
        "logtype": logType,
        "hashcode": hashCode,
        "gamemode": gameMode,
        "user": user
    };
    let lyricsResp = null;
    try {
        lyricsResp = await RequestLyrics(reqBody);
    } catch (error) {
        lyricsResp = error;
    }
    if (lyricsResp === null) return {"status": 500, "statusText": "Internal server error"};

    if (lyricsResp.status == 404) {
        return await GetLyricsData();
    }
    return lyricsResp;
}
```

Funzione 7: Popolamento della pagina game.html

```
async function UpdateLyricsContainer() {
    let lyricsBox = document.getElementById("lyrics-box");
    let copyrightBox = document.getElementById("copyright-box");

    let newLyricsData = await GetLyricsData();
    if (newLyricsData.status != 200) {
        lyricsBox.innerHTML = "<p>" + newLyricsData.status + "</p><p>" + newLyricsData.statusText + "</p>";
        copyrightBox.innerHTML = "Something went wrong";
        return;
    }
    lyricsBox.innerHTML = GetLyricsHTML(newLyricsData.content.lyrics);
    copyrightBox.innerHTML = newLyricsData.content.lyrics_copyright;

    let tracking = document.createElement("img");
    tracking.setAttribute("src", newLyricsData.content.pixel_tracking_url);
    copyrightBox.appendChild(tracking);
}
```

4.1.5 Node.js

Ecco alcune delle funzionalità implementate dal server di backend.

Funzione 8: Endpoint per la registrazione

```
app.put("/register", (req, res) => {
    console.log("Received a register request from origin " + req.hostname);

    if (validateOrigin(req)){
        let password = req.body.password || undefined;
        let username = req.body.username || undefined;

        if (username === undefined || password === undefined) {
            res.json(makeErrorResponseBadRequest());
        } else {
            let check = CheckUsernameAvailability(username);
            if (!check) {
                res.json(makeErrorResponseConflict());
            } else {
                registerData = RegisterUser(username, password);
                console.log("Registered user: " + username);
                res.json(makeOKResponse(registerData));
            }
        }
    } else {
        console.log("Origin not validated");
        res.json(makeErrorResponseForbidden());
    }
});
```

Funzione 9: Endpoint per il login

```
app.post("/login", (req, res) => {
  console.log("Received a login request from origin " + req.hostname);

  if (validateOrigin(req)) {
    let password = req.body.password || undefined;
    let username = req.body.username || undefined;

    if (username === undefined || password === undefined) {
      res.json(makeErrorResponseBadRequest());
    } else {
      if (!fs.existsSync("./users/usernames/" + username)) {
        res.json(makeErrorResponseUnauthorized());
      } else {
        const userData = JSON.parse(fs.readFileSync("./users/usernames/" + username + "/info.json"));

        let hashedPass = userData.hashedPassword;

        let passwordSHA256Hasher = crypto.createHmac("sha256", process.env.SALT);
        let hashedPassword = passwordSHA256Hasher.update(password).digest("base64");

        if (hashedPass == hashedPassword) {
          let responseData = {
            "username": userData.username,
            "hashcode": userData.hashCode
          }
          res.json(makeOKResponse(responseData));
        } else {
          res.json(makeErrorResponseUnauthorized());
        }
      }
    }
  } else {
    console.log("Origin not validated");
    res.json(makeErrorResponseForbidden());
  }
});
```

5. Conclusioni

5.1 Complessità totale del progetto

File HTML e CSS

Il sito possiede un totale di 12 pagine HTML con oltre 1200 righe di CSS.

File Javascript

Il codice Javascript lato client è contenuto all'interno di 6 file per un totale di circa 800 righe per la gestione di chiamate AJAX e controllo del view.

Javascript lato server

Il server di backend è implementato utilizzando NodeJS e uno script di oltre 1100 righe.

Vengono utilizzati diversi moduli, quali:

- Fs
- Express
- Cors
- Axios
- Crypto
- Dotenv
- Body-parser
- String-similarity

5.2

Possibilità di sviluppo

Leaderboard

Un'aggiunta che viene naturale sarebbe una leaderboard tra i diversi giocatori elencando username, punteggio totale e metodo di login.

Oppure si potrebbe comporre due classifiche, una per ciascun metodo di login.

Database

Ampliando le informazioni memorizzate lato server, occorrerebbe implementare l'interazione con un database.

Con la richiesta di una e-mail in fase di registrazione, si potrebbe implementare un metodo di recupero delle credenziali.

Multigiocatore

Al fine di migliorare l'engagement e la qualità del servizio, si potrebbe implementare una modalità multigiocatore (per esempio 1v1).

Portabilità

Migliorare la piattaforma al fine di ottimizzare l'utilizzo su qualsiasi dispositivo.

6. Note

6.1 Sitografia

Documentazione:

<https://www.w3schools.com>

<https://stackoverflow.com>

<https://developer.spotify.com/documentation/web-api/>

<https://developer.musixmatch.com/>

Stile:

<https://coolors.co>

<https://unsplash.com>

<https://fontawesome.com>

6.2 Crediti e ringraziamenti

Si ringraziano i contribuenti al progetto:

- Lisa Stievani:

creatrice del logo.

<https://www.artstation.com/lisastievani>

- Paweł Czerwinski:

creatore delle immagini utilizzate prese da <https://unsplash.com>.

https://unsplash.com/@pawel_czerwinski