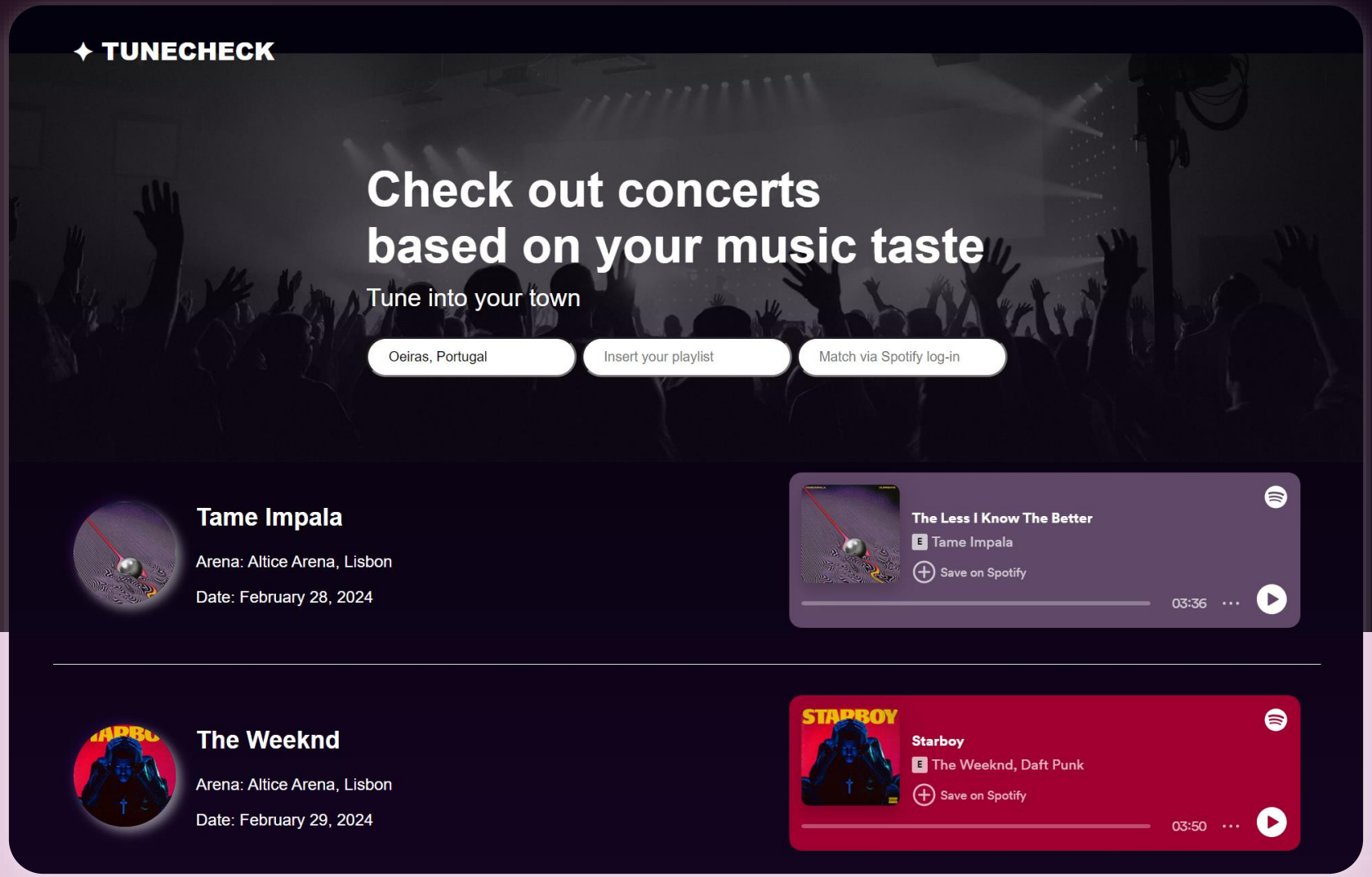
✦ TUNECHECK

Find concerts near you based on your music taste

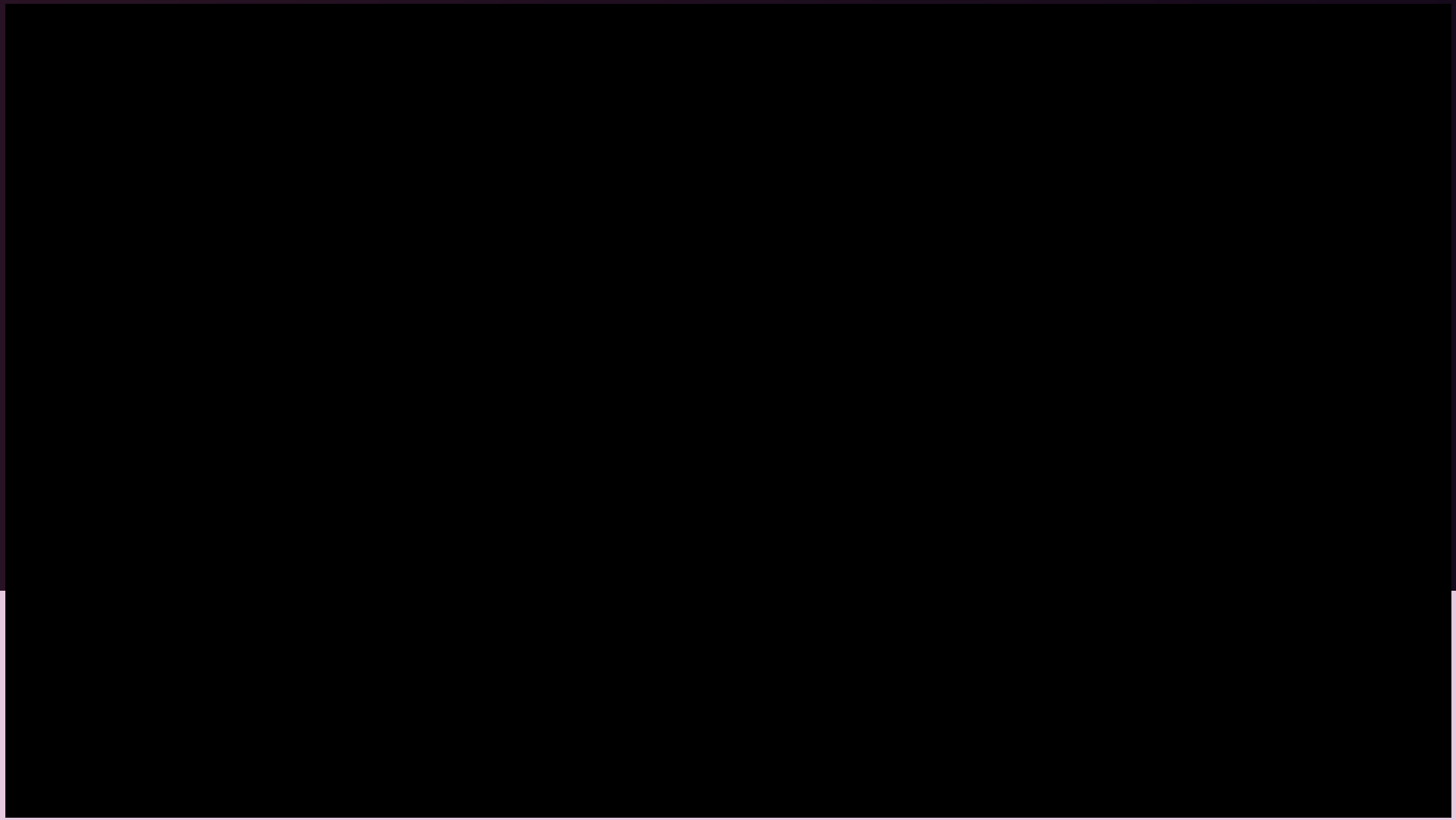
Product | Code

52856_53258_57926_59760

This is what we promised...



...and this is what we delivered!



```google_drive_ref_1.gif
app.get('/login', (req, res) => {
    const SCOPES = 'user-read-private user-read-email user-top-read'; // Include additional scopes as needed
    res.redirect(`https://accounts.spotify.com/authorize?${querystring.stringify({
        response_type: 'code',
        client_id: CLIENT_ID,
        scope: SCOPES,
        redirect_uri: REDIRECT_URI,
        show_dialog: true, // Force the dialog to show on login
    })}`);
});


app.get('/callback', async (req, res) => {
    const code = req.query.code || null;

    try {
        const tokenResponse = await axios({
            method: 'post',
            url: 'https://accounts.spotify.com/api/token',
            data: querystring.stringify({
                grant_type: 'authorization_code',
                code: code,
                redirect_uri: REDIRECT_URI,
            }),
            headers: {
                'Content-Type': 'application/x-www-form-urlencoded',
                'Authorization': `Basic ${Buffer.from(`${CLIENT_ID}:${CLIENT_SECRET}`).toString('base64')}`,
            },
        });

        accessToken = tokenResponse.data.access_token; // Store access token

        // Redirect to home page after login
        res.redirect('/');

    } catch (error) {
        console.error('Access Token Error', error.message);
        res.send('Authentication failed');
    }
});
```

How did we do it?

Spotify

- **Authentication:**
  - OAuth 2.0 authentication flow
  - Request user permission via scopes
  - Redirection to Spotify Login page
- **Callback:**
  - Retrieving authorization code
  - Exchange code for access token
  - Handling access token for API calls

```javascript
app.get('/api/top-artists', async (req, res) => {
    if (!accessToken) {
        return res.status(401).json({ error: 'Not authenticated' });
    }

    try {
        // Fetch top artists
        const artistResponse = await axios.get('https://api.spotify.com/v1/me/top/artists?limit=5', {
            headers: { 'Authorization': `Bearer ${accessToken}` },
        });

        // Extract artist IDs
        const artistIds = artistResponse.data.items.map(artist => artist.id);

        // Fetch genres for these artists (consider batching if >50 artists)
        const genreDict = await getArtistGenreFromArtists(artistIds);

        // Return both artists and genres
        res.json({
            artists: artistResponse.data.items.map(artist => artist.name),
            genres: genreDict,
        });
    } catch (error) {
        console.error("Error:", error.response ? error.response.data : error.message);
        res.status(500).json({ error: 'Failed to fetch data', details: error.response ? error.response.data : error.message });
    }
});
```

Spotify

- **Data Retrieval:**
  - Uses access token to fetch top artists
  - Processes JSON data for artist names and IDs
- **Artists Retrieval:**
  - Calls Spotify API's /me/top/artists endpoint

```
app.get('/api/top-artists', async (req, res) => {
    if (!accessToken) {
        return res.status(401).json({ error: 'Not authenticated' });
    }

    try {
        // Fetch top artists
        // Fetch genres for these artists (consider batching if >50 artists)
        const genreDict = await getArtistGenreFromArtists(artistIds);

        // Return both artists and genres
        res.json({
            artists: artistResponse.data.items.map(artist => artist.name),
            genres: genreDict,
        });
        genres: genreDict,
        });
    } catch (error) {
        console.error("Error:", error.response ? error.response.data : error.message);
        res.status(500).json({ error: 'Failed to fetch data', details: error.response ? error.response.data : error.message });
    }
});
```

Spotify

- **Genre Retrieval:**
  - Maps over artist items to extract names.
  - Captures genres associated with user's top artists

Ticketmaster

- **Event Fetching:**
  - Find Events using top-genres
  - Clean & concatenate Event information
- **Create endpoint:**
  - Data access from client-side

```javascript
async function fetchEventsByGenre(genre, cityInput = userLocation) {
    const startDate = new Date();
    const endDate = new Date();
    const size = 2; // Limit to 2 events to manage load
    endDate.setMonth(endDate.getMonth() + 6); // Set the end date to 6 months from today

    const startDateISO = startDate.toISOString().split('T')[0];
    const endDateISO = endDate.toISOString().split('T')[0];

    const apiKey = getNextTicketmasterApiKey(); // Use the rotated API key
    const url = `https://app.ticketmaster.com/discovery/v2/events.json?apikey=${apiKey}
    &keyword=${genre}&city=${cityInput}&startDateTime=${startDateISO}T00:00:00Z
    &endDateTime=${endDateISO}T23:59:59Z&size=${size}`;


app.get('/api/events', async (req, res) => {
    try {
        const userLocation = req.query.location;
        const genres = await fetchTopGenresFromSpotify(accessToken);
        const eventsPromises = genres.map(async genre => {await delay(1000); return
            fetchEventsByGenre(genre, userLocation);});
        let eventsArrays = await Promise.all(eventsPromises);
        let allEvents = [].concat(...eventsArrays);
        let uniqueEvents = deduplicateEvents(allEvents);

        // Fetch Spotify ID for each event artist and add to event object
        const eventsWithSpotifyIds = await Promise.all(uniqueEvents.map(async event => {
            const spotifyId = await fetchSpotifyArtistId(event.name);
            return { ...event, spotifyId }; // Append Spotify ID to the event object
        }));

        res.json(eventsWithSpotifyIds);
```

```javascript
function fetchAndDisplayEvents(locationInput = globalLocation) {
    const cacheKey = encodeURIComponent(locationInput);
    const cacheExpirationMs = 5 * 60 * 1000; // Cache expiration time (e.g., 5 minutes)
    globalLocation = locationInput;
    // Check if cached data exists and hasn't expired
    if (eventsCache[cacheKey] && (Date.now() - eventsCache[cacheKey].timestamp) < cacheExpirationMs) {
        console.log('Using cached data for:', locationInput);
        const cachedEvents = eventsCache[cacheKey].data;
        originalEvents = [...cachedEvents];
        globalEvents = cachedEvents;
        displayEvents(globalEvents);
        return;
    }

    // Fetch data if no valid cache is found
    fetch(`/api/events?location=${cacheKey}`, {
        method: 'GET',
    })
    .then(response => response.json())
    .then(events => {
        // Update cache with new data
        eventsCache[cacheKey] = {
            timestamp: Date.now(),
            data: events,
        };

        originalEvents = [...events];
        globalEvents = events;
        displayEvents(globalEvents);
    })
    .catch(error => console.error('Error fetching events:', error));
}
```

Populating event data

- **Fetching events:**
  - Using location data
- **Caching:**
  - Saves information
  - Less requests to the API

```javascript
function fetchAndDisplayEvents(locationInput = globalLocation) {
    const cacheKey = encodeURIComponent(locationInput);
    const cacheExpirationMs = 5 * 60 * 1000; // Cache expiration time (e.g., 5 minutes)
    globalLocation = locationInput;
    // Check if cached data exists and hasn't expired
    if (eventsCache[cacheKey] && (Date.now() - eventsCache[cacheKey].timestamp) < cacheExpirationMs) {
        console.log('Using cached data for:', locationInput);
        const cachedEvents = eventsCache[cacheKey].data;
        originalEvents = [...cachedEvents];
        globalEvents = cachedEvents;
        displayEvents(globalEvents);
```

```javascript
function sortAndDisplayEventsByDate() {
    globalEvents.sort((a, b) => new Date(a.startDate) - new Date(b.startDate));
    displayEvents(globalEvents); // Reuse the display logic
}

function displayOriginalOrder() {
    displayEvents(originalEvents); // Display events in the original order
}
```

```javascript
        eventsCache[cacheKey] = {
            timestamp: Date.now(),
            data: events,
        };

        originalEvents = [...events];
        globalEvents = events;
        displayEvents(globalEvents);
    })
    .catch(error => console.error('Error fetching events:', error));
}
```

Dynamic displaying

- **Global variables:**
  - Assigned when fetching events
  - Buttons call different ordering with a display function

AI disclaimer

To complete our assignment, we used generative AI to get recommendations for the codes, implement design elements and resolve issues in the project.