```
import axios from 'axios';

// Spotify API URLs
const BASE_URL = 'https://api.spotify.com/v1';
const AUTH_URL = 'https://accounts.spotify.com/authorize';
const TOKEN_URL = 'https://accounts.spotify.com/api/token';

// Client credentials
const CLIENT_ID = '9c2486f9c4074794909af6be8e8092bd';
const CLIENT_SECRET = '38d294c65b334047a8817a3c5f0de758';
// Use the public URL for the redirect
const REDIRECT_URI = 'http://127.0.0.1:3000/callback';
const SCOPES = [
  'user-read-private',
  'user-read-email',
  'user-library-read',
  'user-library-modify',
  'user-read-playback-state',
  'user-top-read',
  'user-read-recently-played',
  'user-modify-playback-state',
  'user-read-currently-playing',
  'playlist-read-private',
  'playlist-modify-public',
  'playlist-modify-private',
  'user-follow-read'
].join(' ');

// Local storage keys
const TOKEN_KEY = 'spotify_access_token';
const REFRESH_TOKEN_KEY = 'spotify_refresh_token';
const EXPIRATION_KEY = 'spotify_token_expiration';

/**
 * Generates the Spotify authorization URL
 * @returns {string} - Authorization URL
 */
export const getAuthUrl = () => {
  return `${AUTH_URL}?
client_id=${CLIENT_ID}&response_type=code&redirect_uri=${REDIRECT_URI}&scope=${SCO
PES}`;
};

/**
 * Exchanges the authorization code for an access token
 * @param {string} code - Authorization code
 * @returns {Promise} - The response with the tokens
 */
export const getAccessToken = async (code) => {
  try {
    const response = await axios.post(
      TOKEN_URL,
```

```
      new URLSearchParams({
        grant_type: 'authorization_code',
        code,
        redirect_uri: REDIRECT_URI,
        client_id: CLIENT_ID,
        client_secret: CLIENT_SECRET
      }),
      {
        headers: {
          'Content-Type': 'application/x-www-form-urlencoded'
        }
      }
    );

    const expirationTime = new Date().getTime() + response.data.expires_in * 1000;

    // Save tokens to local storage
    localStorage.setItem(TOKEN_KEY, response.data.access_token);
    localStorage.setItem(REFRESH_TOKEN_KEY, response.data.refresh_token);
    localStorage.setItem(EXPIRATION_KEY, expirationTime);

    // After successful login, fetch the user data and save to localStorage
    const userProfile = await getCurrentUserProfile(); // You may already have
this function for user profile
    localStorage.setItem('user', JSON.stringify(userProfile)); // Save user data
to localStorage

    return response.data;
  } catch (error) {
    console.error('Error exchanging code for token:', error);
    throw error;
  }
};


/**
 * Refreshes the access token
 * @returns {Promise} - The response with the new access token
 */
export const refreshToken = async () => {
  const refreshToken = localStorage.getItem(REFRESH_TOKEN_KEY);

  if (!refreshToken) {
    throw new Error('No refresh token available');
  }

  try {
    // Again, this should be done on a server in production
    const response = await axios.post(
      TOKEN_URL,
      new URLSearchParams({
        grant_type: 'refresh_token',
        refresh_token: refreshToken,
        client_id: CLIENT_ID,
```

```
        client_secret: CLIENT_SECRET
      }),
      {
        headers: {
          'Content-Type': 'application/x-www-form-urlencoded'
        }
      }
    );

    const expirationTime = new Date().getTime() + response.data.expires_in * 1000;

    // Update tokens in storage
    localStorage.setItem(TOKEN_KEY, response.data.access_token);
    localStorage.setItem(EXPIRATION_KEY, expirationTime);

    return response.data;
  } catch (error) {
    console.error('Error refreshing token:', error);
    throw error;
  }
};

/**
 * Checks if the current token is valid, refreshes if needed
 * @returns {Promise<string|null>} - Valid access token or null
 */
export const getValidToken = async () => {
  const accessToken = localStorage.getItem(TOKEN_KEY);
  const expirationTime = localStorage.getItem(EXPIRATION_KEY);

  if (!accessToken || !expirationTime) {
    return null;
  }

  // If token expires in less than 5 minutes, refresh it
  if (new Date().getTime() > (expirationTime - 300000)) {
    try {
      const data = await refreshToken();
      return data.access_token;
    } catch (error) {
      console.error('Failed to refresh token:', error);
      return null;
    }
  }

  return accessToken;
};

/**
 * Creates an axios instance with authentication headers
 * @returns {Promise<axios>} - Axios instance
 */
export const getAuthenticatedApi = async () => {
  const token = await getValidToken();
```

```
  if (!token) {
    throw new Error('No valid token available');
  }

  return axios.create({
    baseURL: BASE_URL,
    headers: {
      'Authorization': `Bearer ${token}`
    }
  });
};

/**
 * Gets the current user's profile
 * @returns {Promise} - User profile data
 */
export const getCurrentUserProfile = async () => {
  try {
    const api = await getAuthenticatedApi();
    const response = await api.get('/me');
    return response.data;
  } catch (error) {
    console.error('Error fetching user profile:', error);
    throw error;
  }
};

/**
 * Gets the current user's playlists
 * @returns {Promise} - User playlists
 */
export const getUserPlaylists = async () => {
  try {
    const api = await getAuthenticatedApi();
    const response = await api.get('/me/playlists');
    return response.data.items;
  } catch (error) {
    console.error('Error fetching user playlists:', error);
    throw error;
  }
};
export const getAlbumDetails = async (id) => {
  const api = await getAuthenticatedApi();
  const response = await api.get(`/albums/${id}`);
  return response.data;
};
/**
 * Gets a specific playlist
 * @param {string} playlistId - Playlist ID
 * @returns {Promise} - Playlist details
 */
export const getPlaylist = async (playlistId) => {
  try {
```

```
    const api = await getAuthenticatedApi();
    const response = await api.get(`/playlists/${playlistId}`);
    return response.data;
  } catch (error) {
    console.error(`Error fetching playlist ${playlistId}:`, error);
    throw error;
  }
};

/**
 * Creates a new playlist
 * @param {string} userId - User ID
 * @param {string} name - Playlist name
 * @param {string} description - Playlist description
 * @param {boolean} isPublic - Whether the playlist is public
 * @returns {Promise} - Created playlist
 */
export const createPlaylist = async (userId, name, description, isPublic = true)
=> {
  try {
    const api = await getAuthenticatedApi();
    const response = await api.post(`/users/${userId}/playlists`, {
      name,
      description,
      public: isPublic
    });
    return response.data;
  } catch (error) {
    console.error('Error creating playlist:', error);
    throw error;
  }
};

/**
 * Adds tracks to a playlist
 * @param {string} playlistId - Playlist ID
 * @param {string[]} trackUris - Array of track URIs
 * @returns {Promise} - Response
 */
export const addTracksToPlaylist = async (playlistId, trackUris) => {
  try {
    const api = await getAuthenticatedApi();
    const response = await api.post(`/playlists/${playlistId}/tracks`, {
      uris: trackUris
    });
    return response.data;
  } catch (error) {
    console.error('Error adding tracks to playlist:', error);
    throw error;
  }
};

/**
 * Removes tracks from a playlist
```

```
 * @param {string} playlistId - Playlist ID
 * @param {string[]} trackUris - Array of track URIs
 * @returns {Promise} - Response
 */
export const removeTracksFromPlaylist = async (playlistId, trackUris) => {
  try {
    const api = await getAuthenticatedApi();
    const response = await api.delete(`/playlists/${playlistId}/tracks`, {
      data: {
        tracks: trackUris.map(uri => ({ uri }))
      }
    });
    return response.data;
  } catch (error) {
    console.error('Error removing tracks from playlist:', error);
    throw error;
  }
};

/**
 * Searches Spotify
 * @param {string} query - Search query
 * @param {string} type - Type (track,album,artist,playlist)
 * @param {number} limit - Number of results
 * @returns {Promise} - Search results
 */
export const search = async (query, type = 'track,album,artist,playlist', limit =
20) => {
  try {
    const api = await getAuthenticatedApi();
    const response = await api.get('/search', {
      params: {
        q: query,
        type,
        limit
      }
    });
    return response.data;
  } catch (error) {
    console.error('Error searching:', error);
    throw error;
  }
};

/**
 * Gets an artist by ID
 * @param {string} artistId - Artist ID
 * @returns {Promise} - Artist details
 */
export const getArtist = async (artistId) => {
  try {
    const api = await getAuthenticatedApi();
    const response = await api.get(`/artists/${artistId}`);
    return response.data;
```

```
  } catch (error) {
    console.error(`Error fetching artist ${artistId}:`, error);
    throw error;
  }
};

/**
 * Gets an artist's albums
 * @param {string} artistId - Artist ID
 * @returns {Promise} - Artist's albums
 */
export const getArtistAlbums = async (artistId) => {
  try {
    const api = await getAuthenticatedApi();
    const response = await api.get(`/artists/${artistId}/albums`);
    return response.data;
  } catch (error) {
    console.error(`Error fetching albums for artist ${artistId}:`, error);
    throw error;
  }
};

/**
 * Gets an artist's top tracks
 * @param {string} artistId - Artist ID
 * @param {string} market - Market code (e.g., 'FR')
 * @returns {Promise} - Artist's top tracks
 */
export const getArtistTopTracks = async (artistId, market = 'FR') => {
  try {
    const api = await getAuthenticatedApi();
    const response = await api.get(`/artists/${artistId}/top-tracks`, {
      params: { market }
    });
    return response.data;
  } catch (error) {
    console.error(`Error fetching top tracks for artist ${artistId}:`, error);
    throw error;
  }
};

/**
 * Gets related artists
 * @param {string} artistId - Artist ID
 * @returns {Promise} - Related artists
 */
export const getRelatedArtists = async (artistId) => {
  try {
    const api = await getAuthenticatedApi();
    const response = await api.get(`/artists/${artistId}/related-artists`);
    return response.data;
  } catch (error) {
    console.error(`Error fetching related artists for ${artistId}:`, error);
    throw error;
```

```
  }
};

/**
 * Gets an album by ID
 * @param {string} albumId - Album ID
 * @returns {Promise} - Album details
 */
export const getAlbum = async (albumId) => {
  try {
    const api = await getAuthenticatedApi();
    const response = await api.get(`/albums/${albumId}`);
    return response.data;
  } catch (error) {
    console.error(`Error fetching album ${albumId}:`, error);
    throw error;
  }
};

/**
 * Gets a track by ID
 * @param {string} trackId - Track ID
 * @returns {Promise} - Track details
 */
export const getTrack = async (trackId) => {
  try {
    const api = await getAuthenticatedApi();
    const response = await api.get(`/tracks/${trackId}`);
    return response.data;
  } catch (error) {
    console.error(`Error fetching track ${trackId}:`, error);
    throw error;
  }
};

/**
 * Gets recommended tracks based on seed entities
 * @param {Object} seeds - Seed entities
 * @returns {Promise} - Recommended tracks
 */
export const getRecommendations = async ({ seedArtists = [], seedGenres = [],
seedTracks = [], limit = 20 }) => {
  try {
    const api = await getAuthenticatedApi();
    const response = await api.get('/recommendations', {
      params: {
        seed_artists: seedArtists.join(','),
        seed_genres: seedGenres.join(','),
        seed_tracks: seedTracks.join(','),
        limit
      }
    });
    return response.data;
  } catch (error) {
```

```
    console.error('Error fetching recommendations:', error);
    throw error;
  }
};

/**
 * Gets the current user's recently played tracks
 * @returns {Promise} - Recently played tracks
 */
export const getRecentlyPlayedTracks = async () => {
  try {
    const api = await getAuthenticatedApi();
    const response = await api.get('/me/player/recently-played', {
      params: {
        limit: 5
      }
    });
    return response.data;
  } catch (error) {
    console.error('Error fetching recently played tracks:', error);
    throw error;
  }
};

/**
 * Logs the user out by clearing tokens
 */
export const logout = () => {
  localStorage.removeItem(TOKEN_KEY);
  localStorage.removeItem(REFRESH_TOKEN_KEY);
  localStorage.removeItem(EXPIRATION_KEY);
};
// In services/spotify.js
// Add this function to check authentication
export const checkAuth = async () => {
  const token = await getValidToken();
  if (!token) return false;

  try {
    // Simple request to verify token works
    await getCurrentUserProfile();
    return true;
  } catch (error) {
    logout();
    return false;
  }
};

// Modify your API calls to include error handling
const makeApiCall = async (endpoint, params = {}) => {
  const api = await getAuthenticatedApi();
  try {
    const response = await api.get(endpoint, { params });
    return response.data;
```

```
  } catch (error) {
    if (error.response?.status === 401) {
      // Token expired, try to refresh
      await refreshToken();
      return makeApiCall(endpoint, params);
    }
    throw error;
  }
};

// Then update your existing methods to use makeApiCall
export const getFeaturedPlaylists = async () => {
  return makeApiCall('/browse/featured-playlists', {
    limit: 5,
    country: 'FR'
  });
};

export const getRecentlyPlayed = async () => {
  const api = await getAuthenticatedApi();
  const response = await api.get('/me/player/recently-played');
  return response.data;
};
export const getNewReleases = async () => {
  const api = await getAuthenticatedApi();
  const response = await api.get('/browse/new-releases', {
    params: {
      limit: 5,
      country: 'FR'
    }
  });
  return response.data;
};
// In services/spotify.js
export const getArtistDetails = async (id) => {
  const api = await getAuthenticatedApi()
  const response = await api.get(`/artists/${id}`)
  return response.data
};
export const getFollowedArtists = async () => {
  const api = await getAuthenticatedApi();
  const response = await api.get('/me/following', {
    params: {
      type: 'artist',
      limit: 50
    }
  });
  return response.data;
};

export const getUserSavedAlbums = async () => {
  const api = await getAuthenticatedApi();
  const response = await api.get('/me/albums', {
    params: {
```

```
      limit: 50
    }
  });
  return response.data;
};

export const getFollowedPodcasts = async () => {
  const api = await getAuthenticatedApi();
  const response = await api.get('/me/shows', {
    params: {
      limit: 50
    }
  });
  return response.data.items.map(item => item.show);
};
export default {
  getFollowedArtists,
  getFollowedPodcasts,
  getUserSavedAlbums,
  getAuthUrl,
  getAccessToken,
  refreshToken,
  getValidToken,
  getCurrentUserProfile,
  getUserPlaylists,
  getPlaylist,
  createPlaylist,
  addTracksToPlaylist,
  removeTracksFromPlaylist,
  search,
  getArtist,
  getArtistAlbums,
  getArtistTopTracks,
  getRelatedArtists,
  getAlbum,
  getTrack,
  getRecommendations,
  getRecentlyPlayedTracks,
  logout,
  getFeaturedPlaylists,
  getRecentlyPlayed,
  getNewReleases,
  checkAuth,
  makeApiCall,
 getAlbumDetails,
 getArtistDetails
};
```

```
<template>
    <div class="flex items-center justify-center h-full">
      <div v-if="isLoading" class="text-center">
```

```
        <div class="animate-spin rounded-full h-16 w-16 border-t-2 border-b-2
border-green-500 mx-auto mb-4"></div>
        <p class="text-xl">Connexion en cours...</p>
      </div>
      <div v-else-if="error" class="text-center">
        <div class="text-red-500 mb-4">
          <svg class="w-16 h-16 mx-auto" fill="currentColor" viewBox="0 0 24 24">
            <path d="M12 2C6.48 2 2 6.48 2 12s4.48 10 10 10 10-4.48 10-10S17.52 2
12 2zm1 15h-2v-2h2v2zm0-4h-2V7h2v6z"></path>
          </svg>
        </div>
        <h2 class="text-2xl font-bold mb-2">Erreur d'authentification</h2>
        <p class="text-gray-400 mb-4">{{ error }}</p>
        <router-link to="/" class="btn-primary">Retourner à l'accueil</router-
link>
      </div>
    </div>
  </template>

  <script>
  import { getAccessToken } from '../services/spotify';

  export default {
    name: 'Callback',
    data() {
      return {
        isLoading: true,
        error: null
      }
    },
    async mounted() {
      // Get the authorization code from URL
      const urlParams = new URLSearchParams(window.location.search);
      const code = urlParams.get('code');
      const error = urlParams.get('error');

      if (error) {
        this.error = 'Accès refusé. Veuillez autoriser l\'application.';
        this.isLoading = false;
        return;
      }

      if (!code) {
        this.error = 'Aucun code d\'autorisation fourni.';
        this.isLoading = false;
        return;
      }

      try {
        // Exchange code for access token
        await getAccessToken(code);

        // Redirect to home page
        this.$router.push('/');
```

```
        } catch (err) {
          console.error('Error during token exchange:', err);
          this.error = 'Une erreur s\'est produite lors de l\'authentification.';
          this.isLoading = false;
        }
      }
    }
    </script>
```