Documentation for `routes/spotinite.py`

This module contains the endpoints for what we call, the 'Spotinite service'. It provides routes for using the python spotipy library to interact with the Cyanite API. The Cyanite API is a music recommendation service. It uses the Spotify's ID to identify songs and can recommend songs based on a given artist or band name and a track title.

`similar_tracks(query, user=Depends(login_manager), db=Depends(get_db))` `async`

Fetches and returns a list of tracks similar to the specified song and artist.

This endpoint takes a song title and artist as input, retrieves a Spotify ID for the song, and then fetches a list of similar tracks based on that ID. It aims to return 3 similar tracks that are not by the same artist as the input song, if possible. If not enough non-artist matches are found, it will include tracks by the same artist in the response.

Parameters: - query (SpotiniteQuery): The query object containing the title and artist of the song. - user: The current user object, automatically provided by the login_manager dependency. - db: The database session, automatically provided by the get_db dependency.

Returns: - List[SpotiniteResponse]: A list of similar tracks, each represented by a SpotiniteResponse object.

Source code in `routes/spotinite.py`

```python
@router.post("/similar_tracks", response_model=List[SpotiniteResponse],
tags=["spotinite"])
async def similar_tracks(query: SpotiniteQuery,
user=Depends(login_manager), db: Session = Depends(get_db)):
    """
    Fetches and returns a list of tracks similar to the specified song and
artist.

    This endpoint takes a song title and artist as input, retrieves a
Spotify ID for the song,
    and then fetches a list of similar tracks based on that ID. It aims to
return 3 similar tracks
    that are not by the same artist as the input song, if possible. If not
enough non-artist matches
    are found, it will include tracks by the same artist in the response.

    Parameters:
    - query (SpotiniteQuery): The query object containing the title and
artist of the song.
    - user: The current user object, automatically provided by the
login_manager dependency.
    - db: The database session, automatically provided by the get_db
dependency.

    Returns:
    - List[SpotiniteResponse]: A list of similar tracks, each represented
by a SpotiniteResponse object.

    """
    try:
        spotify_id = get_track_id(query.title, query.artist)
        similar_track_ids = fetch_similar_tracks(spotify_id)
    except Exception as e:
        raise HTTPException(status_code=400, detail=str(e))

    # Fetch 15 similar tracks and return the first 3 that are not by the
same artist if possible
    similar_tracks = []
    added_artists = set()
    backup_tracks = []
    for track_id in similar_track_ids:
        track_info = get_track_info(track_id)
        artist_lower = track_info['Artist'].lower()
        if artist_lower != query.artist.lower() and artist_lower not in
added_artists:
            similar_tracks.append(track_info)
            added_artists.add(artist_lower)
        else:
            backup_tracks.append(track_info)
        if len(similar_tracks) == 3:
            break

    if len(similar_tracks) < 3:
        similar_tracks.extend(backup_tracks[:3-len(similar_tracks)])

    return similar_tracks
```