Documentation for `routes/music.py`

This module contains the endpoints for operations on the music database. It provides routes for searching for music by song title and artist. Mosty usefull for the client to be able to browse music jsute like in a file manager.

Endpoints

Endpoint: `/music_library/add`

Adds a new song to the `music_library` table.

- **Parameters**:
  - **query**: AddSongToMusicLibrary object containing the song details to be added.
  - **user**: User object, automatically provided by the [`login_manager`] dependency.
- **Returns**: A message indicating successful addition of the song.

Source code in `routes/music.py` :

```
@router.post("/add", tags=["songs"])
def add_row(query: AddSongToMusicLibrary, user=Depends(login_manager), db: Session =
    try:
        max_id = db.query(func.max(MusicLibrary.id)).scalar()  # Get the maximum id
        if max_id is None: max_id = 0  # If the table is empty, set max_id to 0
        # insert into the table
        stmt = insert(MusicLibrary).values(
            id=max_id + 1,  # Set the id to one more than the current maximum
            filename=query.filename, filepath=query.filepath,
            album_folder=query.album_folder, artist_folder=query.artist_folder,
            filesize=query.filesize, title=query.title, artist=query.artist,
            album=query.album, year=query.year, tracknumber=query.tracknumber,
            genre=query.genre, top_5_genres=query.top_5_genres,
        )
        db.execute(stmt)
        db.commit()
        return {"message": "Row added successfully"}
    finally:
        db.close()
```

Endpoint: `/music_library/count`

Returns the total number of rows in the `music_library` table.

- **Parameters**: None
- **Returns**: An integer representing the total number of rows in the `music_library` table.

Source code in `routes/music.py`:

```
@router.get("/count", tags=["songs"])
def count_rows(db: Session = Depends(get_db)):
    try:
        result = db.execute(text("SELECT COUNT(*) FROM music_library"))
        count = result.scalar()
        return count
    finally:
        db.close()
```

Endpoint: `/music_library/delete/{id}`

Deletes a specific song from the `music_library` table by its ID.

- **Parameters**:
  - **id**: Integer, the ID of the song to delete.
  - **user**: User object, automatically provided by the [`login_manager`] dependency.
- **Returns**: A message indicating successful deletion of the song. Raises a 404 HTTPException if the song is not found.

Source code in `routes/music.py`:

```
@router.delete("/delete/{id}", tags=["songs"])
def delete_row(id: int, user=Depends(login_manager), db: Session = Depends(get_db)):
    try:
        row = db.query(MusicLibrary).get(id)
        if row is None:
            raise HTTPException(status_code=404, detail="Row not found")
        db.delete(row)
        db.commit()
        return {"message": "Row deleted successfully"}
    finally:
        db.close()
```

Endpoint: `/music_library/album_folder_by_artist_and_album`

Retrieves the album folder for a specific artist and album combination in the `music_library` table.

- **Parameters**:
  - **query**: ArtistAlbumResponse object containing the artist's name and album title.

    - **user**: User object, automatically provided by the [`login_manager`] dependency.
- **Returns**: A dictionary containing the album folder name. Raises a 404 HTTPException if the album is not found.

Source code in `routes/music.py` :

```
@router.post("/album_folder_by_artist_and_album", tags=["songs"])
def get_album_folder_by_artist_and_album(query: ArtistAlbumResponse, user=Depends(lo
    artist = query.artist
    album = query.album
    try:
        row = db.query(MusicLibrary.album_folder).filter(MusicLibrary.artist == arti
        if row is None:
            raise HTTPException(status_code=404, detail="Album not found")
        return {"album_folder": row.album_folder}
    finally:
        db.close()
```

Endpoint: `/music_library/random`

Retrieves a random song from the [`music_library`]table.

- **Parameters**:
    - **user**: User object, automatically provided by the [`login_manager`] dependency.
- **Returns**: A dictionary containing the ID of the randomly selected song and the song's row data.

Source code in `routes/music.py` :

```
@router.get("/random", tags=["songs"])
def get_random_row(user=Depends(login_manager), db: Session = Depends(get_db)):
    try:
        count = db.query(MusicLibrary).count()
        random_id = randint(1, count)
        row = db.query(MusicLibrary).filter(MusicLibrary.id == random_id).first()
        return {"id": random_id, "row": row}
    finally:
        db.close()
```

Endpoint: `/music_library/song/{id}`

Fetches a specific song from the [`music_library`] table by its ID.

- **Parameters**:
    - **id**: Integer, the ID of the song to retrieve.

    - **user**: User object, automatically provided by the [`login_manager`] dependency.
- **Returns**: A dictionary containing the ID of the song and the song's row data. Raises a 404 HTTPException if the song is not found.

Source code in `routes/music.py` :

```
@router.get("/song/{id}", tags=["songs"])
def get_song_by_id(id: int, user=Depends(login_manager), db: Session = Depends(get_d
    try:
        row = db.query(MusicLibrary).filter(MusicLibrary.id == id).first()
        if row is None:
            raise HTTPException(status_code=404, detail="Song not found")
        return {"id": id, "row": row}
    finally:
        db.close()
```

Endpoint: `/music_library/albums`

Lists all albums in the [`music_library`] table, ordered by release date.

- **Parameters**: None
- **Returns**: A list of dictionaries, each containing the album name, album folder, and release year, ordered by release year.

Source code in `routes/music.py` :

```
@router.get("/albums", tags=["songs"])
def list_all_albums(user=Depends(login_manager), db: Session = Depends(get_db)):
    try:
        query = (
            db.query(MusicLibrary.album, MusicLibrary.album_folder, MusicLibrary.yea
            .distinct()
            .order_by(MusicLibrary.year.asc())
        )
        return [{"album": row.album, "album_folder": row.album_folder} for row in qu
    finally:
        db.close()
```

Endpoint: `/music_library/artists`

Lists all artists in the [`music_library`] table in alphabetical order.

- **Parameters**: None
- **Returns**: A list of artist names in alphabetical order.

Source code in `routes/music.py` :

```
@router.get("/artists", tags=["songs"])
def list_all_artists(user=Depends(login_manager), db: Session = Depends(get_db)):
    try:
        query = db.query(MusicLibrary.artist_folder).distinct().order_by(MusicLibrar
        return [row.artist_folder for row in query.all()]
    finally:
        db.close()
```

Endpoint: `/music_library/songs`

Lists all songs from a specific album in the [`music_library`] table.

- **Parameters**:
  - **album_folder**: AlbumResponse object containing the album's folder name.
  - **user**: User object, automatically provided by the [`login_manager`] dependency.
- **Returns**: A list of dictionaries, each containing the track number and title of a song from the specified album.

Source code in `routes/music.py` :

```
@router.post("/songs", tags=["songs"])
def list_all_songs_from_album(album_folder: AlbumResponse = None, user=Depends(logir
    if album_folder is None or album_folder.album_folder is None:
        raise HTTPException(status_code=400, detail="Missing album_folder parameter'
    try:
        query = db.query(MusicLibrary).filter(MusicLibrary.album_folder == album_fol
        return [
            {"tracknumber": row.tracknumber, "title": row.title}
            for row in query.order_by(MusicLibrary.tracknumber.asc()).all()
        ]
    finally:
        db.close()
```

Endpoint: `/music_library/songs/by_artist_and_album`

Lists all songs by a specific artist and from a specific album in the [`music_library`]

- **Parameters**:
  - **query**: ArtistAlbumResponse object containing the artist's name and album title.
  - **user**: User object, automatically provided by the [`login_manager`] dependency.
- **Returns**: A list of dictionaries, each containing the track number, file path, and title of a song from the specified artist and album.

Source code in `routes/music.py` :

```
@router.post("/songs/by_artist_and_album", tags=["songs"])
def list_all_songs_from_artist_and_album(query: ArtistAlbumResponse, user=Depends(lo
    artist = query.artist
    album = query.album
    try:
        query = db.query(MusicLibrary).filter(MusicLibrary.artist == artist, MusicLi
        return [
            {"tracknumber": row.tracknumber, "path": row.filepath, "title": row.titl
            for row in query.order_by(MusicLibrary.tracknumber.asc()).all()
        ]
    finally:
        db.close()
```