

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.

```
add_row(query, user=Depends(login_manager),  
db=Depends(get_db))
```

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`

```

69 @router.post("/add", tags=["songs"])
70 def add_row(query: AddSongToMusicLibrary, user=Depends(login_manager), db:
71     Session = Depends(get_db)):
72     """
73     Adds a new song to the music_library table.
74
75     - **Parameters**:
76       - **query**: AddSongToMusicLibrary object containing the song
77       details to be added.
78       - **user**: User object, automatically provided by the
79       login_manager dependency.
80       - **Returns**: A message indicating successful addition of the song.
81     """
82     try:
83         max_id = db.query(func.max(MusicLibrary.id)).scalar() # Get the
84         maximum id from the music_library table
85         if max_id is None: max_id = 0 # If the table is empty, set max_id
86         to 0
87
88         # insert into the table
89         stmt = insert(MusicLibrary).values(
90             id=max_id + 1, # Set the id to one more than the current
91             maximum
92             filename=query.filename, filepath=query.filepath,
93             album_folder=query.album_folder,
94             artist_folder=query.artist_folder, filesize=query.filesize,
95             title=query.title,
96             artist=query.artist, album=query.album, year=query.year,
97             tracknumber=query.tracknumber,
98             genre=query.genre, top_5_genres=query.top_5_genres,
99         )
100         db.execute(stmt)
101         db.commit()
102         return {"message": "Row added successfully"}
103     finally:
104         db.close()

```

`count_rows(db=Depends(get_db))`

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`

```

16 @router.get("/count", tags=["songs"])
17 def count_rows(db: Session = Depends(get_db)):
18     """
19     Returns the total number of rows in the music_library table.
20
21     - **Parameters**: None
22     - **Returns**: An integer representing the total number of rows in the
23     music_library table.
24     """
25     try:
26         result = db.execute(text("SELECT COUNT(*) FROM music_library"))
27         count = result.scalar()
28         return count
29     finally:
30         db.close()

```

```

delete_row(id, user=Depends(login_manager),
db=Depends(get_db))

```

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`

```

98 @router.delete("/{delete}/{id}", tags=["songs"])
99 def delete_row(id: int, user=Depends(login_manager), db: Session =
100 Depends(get_db)):
101     """
102     Deletes a specific song from the music_library table by its ID.
103
104     - **Parameters**:
105         - **id**: Integer, the ID of the song to delete.
106         - **user**: User object, automatically provided by the
107 login_manager dependency.
108     - **Returns**: A message indicating successful deletion of the song.
109     Raises a 404 HTTPException if the song is not found.
110     """
111     try:
112         row = db.query(MusicLibrary).get(id)
113         if row is None:
114             raise HTTPException(status_code=404, detail="Row not found")
115         db.delete(row)
116         db.commit()
117         return {"message": "Row deleted successfully"}
118     finally:
119         db.close()

```

```

get_album_folder_by_artist_and_album(query,
user=Depends(login_manager), db=Depends(get_db))

```

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`

```

222 @router.post("/album_folder_by_artist_and_album", tags=["songs"])
223 def get_album_folder_by_artist_and_album(
224     query: ArtistAlbumResponse, user=Depends(login_manager), db: Session =
225     Depends(get_db)
226 ):
227     """
228     Retrieves the album folder for a specific artist and album combination
229     in the music_library table.
230
231     - **Parameters**:
232       - **query**: ArtistAlbumResponse object containing the artist's
233       name and album title.
234       - **user**: User object, automatically provided by the
235       login_manager dependency.
236       - **Returns**: A dictionary containing the album folder name. Raises a
237       404 HTTPException if the album is not found.
238     """
239     artist = query.artist
240     album = query.album
241     try:
242         row =
db.query(MusicLibrary.album_folder).filter(MusicLibrary.artist == artist,
MusicLibrary.album == album).first()
        if row is None:
            raise HTTPException(status_code=404, detail="Album not found")
        return {"album_folder": row.album_folder}
    finally:
        db.close()

```

```

get_random_row(user=Depends(login_manager),
db=Depends(get_db))

```

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

```

32 @router.get("/random", tags=["songs"])
33 def get_random_row(user=Depends(login_manager), db: Session =
34 Depends(get_db)):
35     """
36     Retrieves a random song from the music_library table.
37
38     - **Parameters**:
39       - **user**: User object, automatically provided by the
40       login_manager dependency.
41       - **Returns**: A dictionary containing the ID of the randomly selected
42       song and the song's row data.
43     """
44     try:
45         count = db.query(MusicLibrary).count()
46         random_id = randint(1, count)
47         row = db.query(MusicLibrary).filter(MusicLibrary.id ==
         random_id).first()
         return {"id": random_id, "row": row}
     finally:
         db.close()

```

get_song_by_id(id, user=Depends(login_manager),
db=Depends(get_db))

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

```

50 @router.get("/song/{id}", tags=["songs"])
51 def get_song_by_id(id: int, user=Depends(login_manager), db: Session =
52     Depends(get_db)):
53     """
54     Fetches a specific song from the music_library table by its ID.
55
56     - **Parameters**:
57         - **id**: Integer, the ID of the song to retrieve.
58         - **user**: User object, automatically provided by the
59         login_manager dependency.
60     - **Returns**: A dictionary containing the ID of the song and the
61     song's row data. Raises a 404 HTTPException if the song is not found.
62     """
63     try:
64         row = db.query(MusicLibrary).filter(MusicLibrary.id == id).first()
65         if row is None:
66             raise HTTPException(status_code=404, detail="Song not found")
67         return {"id": id, "row": row}
68     finally:
69         db.close()

```

```

list_all_albums(user=Depends(login_manager),
db=Depends(get_db))

```

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`

```

203 @router.get("/albums", tags=["songs"])
204 def list_all_albums(user=Depends(login_manager), db: Session =
205     Depends(get_db)):
206     """
207     Lists all albums in the music_library table, ordered by release date.
208
209     - **Parameters**: None
210     - **Returns**: A list of dictionaries, each containing the album name,
211     album folder, and release year, ordered by release year.
212     """
213     try:
214         query = (
215             db.query(MusicLibrary.album, MusicLibrary.album_folder,
216 MusicLibrary.year)
217             .distinct()
218             .order_by(MusicLibrary.year.asc())
219         )
220         return [{"album": row.album, "album_folder": row.album_folder} for
221 row in query.all()]
222     finally:
223         db.close()

```

```

list_all_albums_from_artist(artist_folder,
user=Depends(login_manager), db=Depends(get_db))

```

Lists all albums by a specific artist in the music_library table, ordered by release date.

- **Parameters:**
 - **artist_folder:** ArtistFolderResponse object containing the artist's folder name.
 - **user:** User object, automatically provided by the login_manager dependency.
- **Returns:** A list of album names for the given artist, ordered by release date.

Source code in `routes/music.py`

```

134 @router.post("/albums", tags=["songs"])
135 def list_all_albums_from_artist(artist_folder: ArtistFolderResponse,
136 user=Depends(login_manager), db: Session = Depends(get_db)):
137     """
138     Lists all albums by a specific artist in the music_library table,
139     ordered by release date.
140
141     - **Parameters**:
142       - **artist_folder**: ArtistFolderResponse object containing the
143       artist's folder name.
144       - **user**: User object, automatically provided by the
145       login_manager dependency.
146       - **Returns**: A list of album names for the given artist, ordered by
147       release date.
148     """
149     if artist_folder is None or artist_folder.artist_folder is None:
150         raise HTTPException(status_code=400, detail="Missing artist_folder
151 parameter")
152     try:
153         query = (
154             db.query(MusicLibrary.album)
155             .filter(MusicLibrary.artist_folder ==
156                   artist_folder.artist_folder)
157             .distinct()
158         )
159         return [row.album for row in query.all()]
160     finally:
161         db.close()

```

```
list_all_artists(user=Depends(login_manager),
db=Depends(get_db))
```

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`

```

119 @router.get("/artists", tags=["songs"])
120 def list_all_artists(user=Depends(login_manager), db: Session =
121     Depends(get_db)):
122     """
123     Lists all artists in the music_library table in alphabetical order.
124
125     - **Parameters**: None
126     - **Returns**: A list of artist names in alphabetical order.
127     """
128     try:
129         query =
130     (db.query(MusicLibrary.artist_folder).distinct().order_by(MusicLibrary.artis
131         return [row.artist_folder for row in query.all()]
132     finally:
133         db.close()

```

```

list_all_songs_from_album(album_folder=None,
user=Depends(login_manager), db=Depends(get_db))

```

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`

```

157 @router.post("/songs", tags=["songs"])
158 def list_all_songs_from_album(album_folder: AlbumResponse = None,
159 user=Depends(login_manager), db: Session = Depends(get_db)):
160     """
161     Lists all songs from a specific album in the music_library table.
162
163     - **Parameters**:
164       - **album_folder**: AlbumResponse object containing the album's
165       folder name.
166       - **user**: User object, automatically provided by the
167       login_manager dependency.
168       - **Returns**: A list of dictionaries, each containing the track
169       number and title of a song from the specified album.
170     """
171     if album_folder is None or album_folder.album_folder is None:
172         raise HTTPException(status_code=400, detail="Missing album_folder
173 parameter")
174     try:
175         query = db.query(MusicLibrary).filter(MusicLibrary.album_folder ==
176 album_folder.album_folder)
177         return [
178             {"tracknumber": row.tracknumber, "title": row.title}
179             for row in
180 query.order_by(MusicLibrary.tracknumber.asc()).all()
181         ]
182     finally:
183         db.close()

```

```
list_all_songs_from_artist_and_album(query,
user=Depends(login_manager), db=Depends(get_db))
```

Lists all songs by a specific artist and from a specific album 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 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

```

179 @router.post("/songs/by_artist_and_album", tags=["songs"])
180 def list_all_songs_from_artist_and_album(
181     query: ArtistAlbumResponse, user=Depends(login_manager), db: Session =
182     Depends(get_db)
183 ):
184     """
185     Lists all songs by a specific artist and from a specific album in the
186     music_library table.
187
188     - **Parameters**:
189         - **query**: ArtistAlbumResponse object containing the artist's
190         name and album title.
191         - **user**: User object, automatically provided by the
192         login_manager dependency.
193         - **Returns**: A list of dictionaries, each containing the track
194         number, file path, and title of a song from the specified artist and
195         album.
196     """
197     artist = query.artist
198     album = query.album
199     try:
200         query = db.query(MusicLibrary).filter(MusicLibrary.artist ==
artist, MusicLibrary.album == album)
        return [
            {"tracknumber": row.tracknumber, "path": row.filepath,
"title": row.title}
            for row in
query.order_by(MusicLibrary.tracknumber.asc()).all()
        ]
    finally:
        db.close()

```