

Documentation for `routes/favorites.py`

This module contains the endpoints for the favorites service. It provides routes for adding, removing, and listing favorite songs for a user.

```
add_song_to_favorites(song,  
user=Depends(login_manager), db=Depends(get_db))  
async
```

Add a song to the authenticated user's list of favorites.

- **song**: SongPath - The path of the song to be added to favorites.
- **user**: User - The authenticated user who is adding the song to favorites.
- **db**: Session - The database session for querying and updating the database.
- **return**: Returns a message indicating the song was successfully added to favorites or if it was already in favorites.

Source code in routes/favorites.py

```

35 @router.post("/add", tags=["favorites"])
36 async def add_song_to_favorites(song: SongPath, user: User =
37 Depends(login_manager), db: Session = Depends(get_db)):
38     """
39     Add a song to the authenticated user's list of favorites.
40
41     - **song**: SongPath - The path of the song to be added to favorites.
42     - **user**: User - The authenticated user who is adding the song to
43     favorites.
44     - **db**: Session - The database session for querying and updating the
45     database.
46     - **return**: Returns a message indicating the song was successfully
47     added to favorites or if it was already in favorites.
48     """
49     user = db.merge(user)
50     db.refresh(user)
51
52     if len(user.favorites) >= 9:
53         # Remove the oldest song from the favorites
54         user.favorites.pop(0)
55
56     music_id = get_song_id_by_filepath(db, song.file_path)
57     if not music_id:
58         raise HTTPException(status_code=404, detail="Song not found")
59     music = db.query(MusicLibrary).get(music_id)
60
61     # Check if the song is already in the user's favorites
62     if music in user.favorites:
63         return {"message": "Song is already in favorites"}
64
65     user.favorites.append(music)
66     db.commit()
67     return {"message": "Song added to favorites"}

```

```

delete_song_from_favorites(song,
user=Depends(login_manager), db=Depends(get_db))
async

```

Remove a song from the authenticated user's list of favorites.

- **song**: SongPath - The path of the song to be removed from favorites.
- **user**: User - The authenticated user who is removing the song from favorites.
- **db**: Session - The database session for querying and updating the database.
- **return**: Returns a message indicating the song was successfully removed from favorites or if the song was not found in favorites.

Source code in `routes/favorites.py`

```

66 @router.delete("/delete", tags=["favorites"])
67 async def delete_song_from_favorites(song: SongPath, user: User =
68 Depends(login_manager), db: Session = Depends(get_db)):
69     """
70     Remove a song from the authenticated user's list of favorites.
71
72     - **song**: SongPath - The path of the song to be removed from
73     favorites.
74     - **user**: User - The authenticated user who is removing the song from
75     favorites.
76     - **db**: Session - The database session for querying and updating the
77     database.
78     - **return**: Returns a message indicating the song was successfully
79     removed from favorites or if the song was not found in favorites.
80     """
81     user = db.merge(user)
82     db.refresh(user)
83     music_id = get_song_id_by_filepath(db, song.file_path)
84     if not music_id:
85         raise HTTPException(status_code=404, detail="Song not found")
86     music = db.query(MusicLibrary).get(music_id)
87     for favorite in user.favorites:
88         if favorite.id == music.id:
89             user.favorites.remove(favorite)
90             db.commit()
91             return {"message": "Song removed from favorites"}
92     raise HTTPException(status_code=404, detail="Song not found in
93     favorites")

```

`get_favorites(user=Depends(login_manager),`
`db=Depends(get_db))` `async`

Retrieve the list of favorite songs for the authenticated user.

- **user:** User - The authenticated user whose favorites are to be retrieved.
- **db:** Session - The database session for querying the database.
- **return:** Returns a list of the user's favorite songs.

Source code in `routes/favorites.py`

```

15 @router.get("/", tags=["favorites"])
16 async def get_favorites(user=Depends(login_manager), db: Session =
17 Depends(get_db)):
18     """
19     Retrieve the list of favorite songs for the authenticated user.
20
21     - **user**: User - The authenticated user whose favorites are to be
22     retrieved.
23     - **db**: Session - The database session for querying the database.
24     - **return**: Returns a list of the user's favorite songs.
25     """
26     # The merge() function is used to merge a detached object back into the
27     session.
28     # It returns a new instance that represents the existing row in the DB.
29     # This is necessary because the 'user' object might have been created
30     in a different session and we want to associate it with the current
31     session.
32     user = db.merge(user)
33
34     # The refresh() function is used to update the attributes of the 'user'
35     instance with the current data in the DB.
36     # This is necessary because the 'user' object might have stale data and
37     we want to ensure we're working with the most recent data.
38     db.refresh(user)
39     return user.favorites

```