

Documentation for `routes/minio.py`

This module contains the endpoints for the MiniO service. It provides endpoints for storing and retrieving objects from MiniO buckets.

`download_file(query, user=Depends(login_manager))`

`async`

Downloads a song file from MinIO storage.

- **query:** SongPath - The path to the song file in MinIO storage.
- **user:** User - The authenticated user making the request.
- **return:** StreamingResponse - A streaming response for downloading the song file.

” Source code in `routes/minio.py`

```

79 @router.post("/download-song/", tags=["MinIO"])
80 async def download_file(query: SongPath, user=Depends(login_manager)):
81     """
82     Downloads a song file from MinIO storage.
83
84     - **query**: SongPath - The path to the song file in MinIO storage.
85     - **user**: User - The authenticated user making the request.
86     - **return**: StreamingResponse - A streaming response for downloading
87     the song file.
88     """
89     try:
90         data = minio_client.get_object(DEFAULT_SETTINGS.minio_bucket_name,
91         query.file_path)
92         filename = query.file_path.split('/')[-1] # Get the filename from
93         the file_path
94         headers = {
95             "Content-Disposition": f"attachment; filename={filename}",
96         }
97         return StreamingResponse(data.stream(32*1024),
98         media_type="audio/mpeg", headers=headers)
99     except Exception as e:
100         raise HTTPException(status_code=404, detail="File not found")

```

`get_file(query, user=Depends(login_manager))` `async`

Streams a song file from MinIO storage.

- **query:** SongPath - The path to the song file in MinIO storage.
- **user:** User - The authenticated user making the request.
- **return:** StreamingResponse - A streaming response of the song file.

” Source code in `routes/minio.py`

```

63 @router.post("/stream-song/", tags=["MinIO"])
64 async def get_file(query: SongPath, user=Depends(login_manager)):
65     """
66     Streams a song file from MinIO storage.
67
68     - **query**: SongPath - The path to the song file in MinIO storage.
69     - **user**: User - The authenticated user making the request.
70     - **return**: StreamingResponse - A streaming response of the song
71     file.
72     """
73     try:
74         data = minio_client.get_object(DEFAULT_SETTINGS.minio_bucket_name,
75         query.file_path)
76         return StreamingResponse(data.stream(32*1024),
77         media_type="audio/mpeg")
78     except Exception as e:
79         raise HTTPException(status_code=404, detail="File not found")

```

```

get_random_song_metadata(user=Depends(login_manager),
db=Depends(get_db)) async

```

Retrieves metadata for a random song from MinIO storage using the music-tag library.

- **user:** User - The authenticated user making the request.
- **db:** Session - Database session dependency.
- **return:** JSONResponse - The metadata of a random song.

Source code in `routes/minio.py`

```

115 @router.get("/random-metadata", tags=["MinIO"])
116 async def get_random_song_metadata(user=Depends(login_manager), db:
117     Session = Depends(get_db)):
118     """
119     Retrieves metadata for a random song from MinIO storage using the
120     music-tag library.
121
122     - **user**: User - The authenticated user making the request.
123     - **db**: Session - Database session dependency.
124     - **return**: JSONResponse - The metadata of a random song.
125     """
126     try:
127         count = db.query(MusicLibrary).count()
128         random_id = randint(1, count)
129         row = db.query(MusicLibrary).filter(MusicLibrary.id ==
130 random_id).first()
131         metadata =
132 get_metadata_and_artwork(DEFAULT_SETTINGS.minio_bucket_name, row.filepath)
133         return JSONResponse(content=metadata)
    except Exception as e:
        raise HTTPException(status_code=400, detail=str(e))
    finally:
        db.close()

```

`get_song_metadata(query, user=Depends(login_manager))`

`async`

Retrieves metadata for a specified song from MinIO storage using the music-tag library.

- **query:** SongPath - The path to the song file in MinIO storage.
- **user:** User - The authenticated user making the request.
- **return:** JSONResponse - The metadata of the specified song.

Source code in `routes/minio.py`

```

99 @router.post("/metadata", tags=["MinIO"])
100 async def get_song_metadata(query: SongPath, user=Depends(login_manager)):
101     """
102     Retrieves metadata for a specified song from MinIO storage using the
103     music-tag library.
104
105     - **query**: SongPath - The path to the song file in MinIO storage.
106     - **user**: User - The authenticated user making the request.
107     - **return**: JSONResponse - The metadata of the specified song.
108     """
109     try:
110         metadata =
111         get_metadata_and_artwork(DEFAULT_SETTINGS.minio_bucket_name,
112         query.file_path)
113         return JSONResponse(content=metadata)
114     except Exception as e:
115         raise HTTPException(status_code=400, detail=str(e))

```

```

list_objects_in_album_folder(query,
user=Depends(login_manager))

```

Retrieves a list of objects within a specified album folder in the MinIO bucket.

- **query:** AlbumResponse - The album folder to list objects from.
- **user:** User - The authenticated user making the request.
- **return:** List[S3Object] - A list of objects found in the specified album folder.

Source code in `routes/minio.py`

```

20 @router.post("/list-objects/", response_model=List[S3Object], tags=
21 ["MinIO"])
22 def list_objects_in_album_folder(query: AlbumResponse,
23 user=Depends(login_manager)):
24     """
25     Retrieves a list of objects within a specified album folder in the
26     MinIO bucket.
27
28     - **query**: AlbumResponse - The album folder to list objects from.
29     - **user**: User - The authenticated user making the request.
30     - **return**: List[S3Object] - A list of objects found in the specified
31     album folder.
32     """
33     objects = minio_client.list_objects(
34         DEFAULT_SETTINGS.minio_bucket_name,
35         prefix=query.album_folder,
36         recursive=True)
37
38     response = []
39     for obj in objects:
40         s3_object = {
41             "name": obj.object_name,
42             "size": obj.size,
43             "etag": obj.etag,
44             "last_modified": obj.last_modified.isoformat()
45         }
46         response.append(s3_object)
47
48     return response

```

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

Lists objects uploaded by the authenticated user.

- **user**: User - The authenticated user making the request.
- **db**: Session - Database session dependency.
- **return**: UploadMP3ResponseList - A list of uploaded objects by the user.

Source code in routes/minio.py

```

47 @router.post("/list-uploaded-objects",
48 response_model=UploadMP3ResponseList, tags=["MinIO"])
49 def list_uploaded_objects(user=Depends(login_manager), db: Session =
50 Depends(get_db)):
51     """
52     Lists objects uploaded by the authenticated user.
53
54     - **user**: User - The authenticated user making the request.
55     - **db**: Session - Database session dependency.
56     - **return**: UploadMP3ResponseList - A list of uploaded objects by the
57     user.
58     """
59     objects =
60     minio_client.list_objects(DEFAULT_SETTINGS.minio_temp_bucket_name)
        # Adjusting the response to match the expected structure
        uploads = [UploadDetail(filename=obj.object_name) for obj in objects]
        response = UploadMP3ResponseList(uploads=uploads)
        return response

```

```

upload_file(file=File(...),
user=Depends(login_manager), db=Depends(get_db))
async

```

Uploads a MP3 file to MinIO storage using a temporary bucket.

- **file:** UploadFile - The MP3 file to upload.
- **user:** User - The authenticated user making the request.
- **db:** Session - Database session dependency.
- **return:** UploadMP3ResponseList - A list of uploaded MP3 files by the user.

Source code in routes/minio.py

```

136 @router.post("/upload-temp", tags=["MinIO"],
137 response_model=UploadMP3ResponseList)
138 async def upload_file(file: UploadFile = File(...),
139 user=Depends(login_manager), db: Session = Depends(get_db)):
140     """
141     Uploads a MP3 file to MinIO storage using a temporary bucket.
142
143     - **file**: UploadFile - The MP3 file to upload.
144     - **user**: User - The authenticated user making the request.
145     - **db**: Session - Database session dependency.
146     - **return**: UploadMP3ResponseList - A list of uploaded MP3 files by
147     the user.
148     """
149     try: # Check content type and extension
150         if file.content_type != "audio/mpeg":
151             raise HTTPException(status_code=400, detail="Only MP3 files
152 are allowed.")
153         _, file_extension = os.path.splitext(file.filename)
154         if file_extension.lower() != ".mp3":
155             raise HTTPException(status_code=400, detail="The uploaded file
156 is not an MP3 file.")
157
158         # Generate a secure filename
159         secure_filename = sanitize_filename(file.filename)
160
161         # Determine the size of the uploaded file by moving the cursor to
162         the end to get the file size
163         file.file.seek(0, os.SEEK_END)
164         file_size = file.file.tell()
165         file.file.seek(0)
166
167         # Stream the file directly to MinIO
168         minio_client.put_object(
169             bucket_name=DEFAULT_SETTINGS.minio_temp_bucket_name,
170             object_name=secure_filename,
171             data=file.file,
172             length=file_size,
173             content_type=file.content_type
174         )
175
176         # Store upload information in the database and return the updated
177         list of uploaded songs by the user
178         # song_path_in_minio = f"
179         {DEFAULT_SETTINGS.minio_temp_bucket_name}/{secure_filename}"
180         store_upload_info(db, user.id, secure_filename)
181         uploaded_songs = get_user_uploads(db, user.id)
182
183         return UploadMP3ResponseList(uploads=uploaded_songs)
184     except Exception as e:
185         raise HTTPException(status_code=500, detail=f"An unexpected error
186 occurred. {str(e)}")

```