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.

Endpoints

Endpoint: `/minio/download-song/`

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`:

```
@router.post("/download-song/", tags=["MinIO"])
async def download_file(query: SongPath, user=Depends(login_manager)):
    try:
        data = minio_client.get_object(DEFAULT_SETTINGS.minio_bucket_name, query.fil
        filename = query.file_path.split('/')[-1]  # Get the filename from the file_
        headers = {
            "Content-Disposition": f"attachment; filename={filename}",
        }
        return StreamingResponse(data.stream(32*1024), media_type="audio/mpeg", heac
    except Exception as e:
        raise HTTPException(status_code=404, detail="File not found")
```

Endpoint: `/minio/stream-song/`

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`:

```
@router.post("/stream-song/", tags=["MinIO"])
async def get_file(query: SongPath, user=Depends(login_manager)):
    try:
        data = minio_client.get_object(DEFAULT_SETTINGS.minio_bucket_name, query.fil
        return StreamingResponse(data.stream(32*1024), media_type="audio/mpeg")
    except Exception as e:
        raise HTTPException(status_code=404, detail="File not found")
```

Endpoint: `/minio/random-metadata`

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`:

```
@router.get("/random-metadata", tags=["MinIO"])
async def get_random_song_metadata(user=Depends(login_manager), db: Session = Depenc
    try:
        count = db.query(MusicLibrary).count()
        random_id = randint(1, count)
        row = db.query(MusicLibrary).filter(MusicLibrary.id == random_id).first()
        metadata = get_metadata_and_artwork(DEFAULT_SETTINGS.minio_bucket_name, row.
        return JSONResponse(content=metadata)
    except Exception as e:
        raise HTTPException(status_code=400, detail=str(e))
    finally:
        db.close()
```

Endpoint: `/minio/metadata`

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`:

```
@router.post("/metadata", tags=["MinIO"])
async def get_song_metadata(query: SongPath, user=Depends(login_manager)):
    try:
        metadata = get_metadata_and_artwork(DEFAULT_SETTINGS.minio_bucket_name, quer
        return JSONResponse(content=metadata)
    except Exception as e:
        raise HTTPException(status_code=400, detail=str(e))
```

Endpoint: `/minio/list-objects/`

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` :

```
@router.post("/list-objects/", response_model=List[S3Object], tags=["MinIO"])
def list_objects_in_album_folder(query: AlbumResponse, user=Depends(login_manager)):
    objects = minio_client.list_objects(
        DEFAULT_SETTINGS.minio_bucket_name,
        prefix=query.album_folder,
        recursive=True
    )
    response = []
    for obj in objects:
        s3_object = {
            "name": obj.object_name,
            "size": obj.size,
            "etag": obj.etag,
            "last_modified": obj.last_modified.isoformat()
        }
        response.append(s3_object)
    return response
```

Endpoint: `/minio/list-uploaded-objects`

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` :

```
@router.post("/list-uploaded-objects", response_model=UploadMP3ResponseList, tags=['
def list_uploaded_objects(user=Depends(login_manager), db: Session = Depends(get_db)
    objects = 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
```

Endpoint: `/minio/upload-temp`

Uploads an 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` :

```
@router.post("/upload-temp", tags=["MinIO"], response_model=UploadMP3ResponseList)
async def upload_file(file: UploadFile = File(...), user=Depends(login_manager), db:
    try:
        # Check content type and extension
        if file.content_type != "audio/mpeg":
            raise HTTPException(status_code=400, detail="Only MP3 files are allowed.
        _, file_extension = os.path.splitext(file.filename)
        if file_extension.lower() != ".mp3":
            raise HTTPException(status_code=400, detail="The uploaded file is not ar
        # Generate a secure filename
        secure_filename = sanitize_filename(file.filename)
        # Determine the size of the uploaded file by moving the cursor to the end tc
        file.file.seek(0, os.SEEK_END)
        file_size = file.file.tell()
        file.file.seek(0)
        # Stream the file directly to MinIO
        minio_client.put_object(
            bucket_name=DEFAULT_SETTINGS.minio_temp_bucket_name,
            object_name=secure_filename,
            data=file.file,
            length=file_size,
            content_type=file.content_type
        )
        # Store upload information in the database and return the updated list of up
        store_upload_info(db, user.id, secure_filename)
        uploaded_songs = get_user_uploads(db, user.id)
        return UploadMP3ResponseList(uploads=uploaded_songs)
    except Exception as e:
        raise HTTPException(status_code=400, detail=str(e))
```

Endpoint: `/minio/delete-temp`

Deletes a MP3 file from MinIO bucket.

- **query**: SongPath - The path to the MP3 file in MinIO storage.
- **user**: User - The authenticated user making the request.
- **db**: Session - Database session dependency.
- **return**: UploadMP3ResponseList - A list of uploaded MP3 files by the user.

For more details, visit the documentation.