UNIVERSITÀ
di VERONA

ELABORATO DI INGEGNERIA DEL SOFTWARE

2024/2025

ShareSound:
Piattaforma per la condivisione di contenuti musicali

MIOLATO ALESSANDRO VR446081
TERZIU FABIO VR471449

INDICE:

1 INTRODUZIONE AL SOFTWARE

1.1 Traccia scelta

ShareSound è un sistema software progettato per gestire la collezione e la condivisione di materiale musicale di vario genere.

Il sistema permette a differenti utenti autorizzati dall'amministratore del sistema di inserire, gestire e condividere informazioni e materiale digitale relativo ai brani musicali, come spartiti, testi, accordi, file mp3 e file mp4.

Permette inoltre il caricamento e la riproduzione di concerti presenti su YouTube, con la possibilità di arricchirli con meta-informazioni relative ad aspetti tecnici.

2 DEFINIZIONE DEI REQUISITI

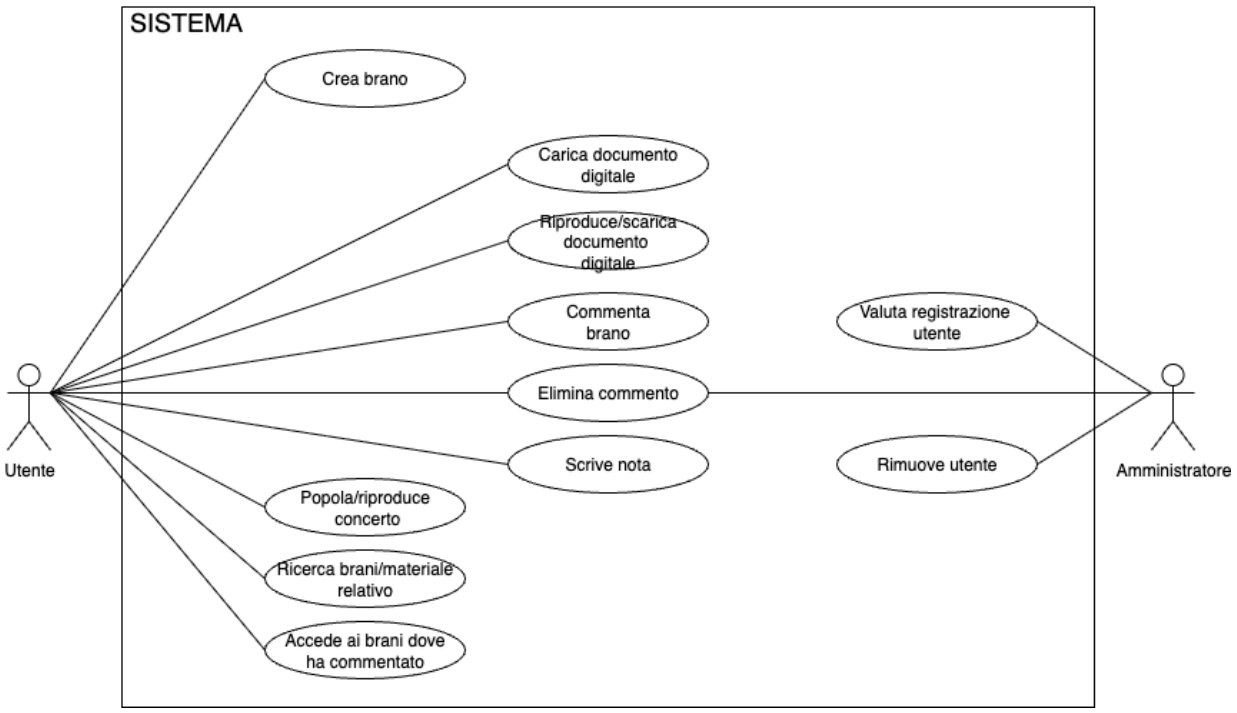
2.1 Use case e sequence diagram

Gli Use-Case sono tra i più importanti formalismi di UML, e permettono di esprimere un comportamento nel sistema.

Permette di evidenziare in modo chiaro e non interpretabile chi (attore) e che cosa esso fa nel sistema (casi d'uso).

Lo use case non considera l'avanzamento del tempo, ma consente solo di descrivere azione e obiettivo, quindi non la sequenza esatta delle procedure.

Possiamo dire che rappresenta una fotografia del nostro sistema.



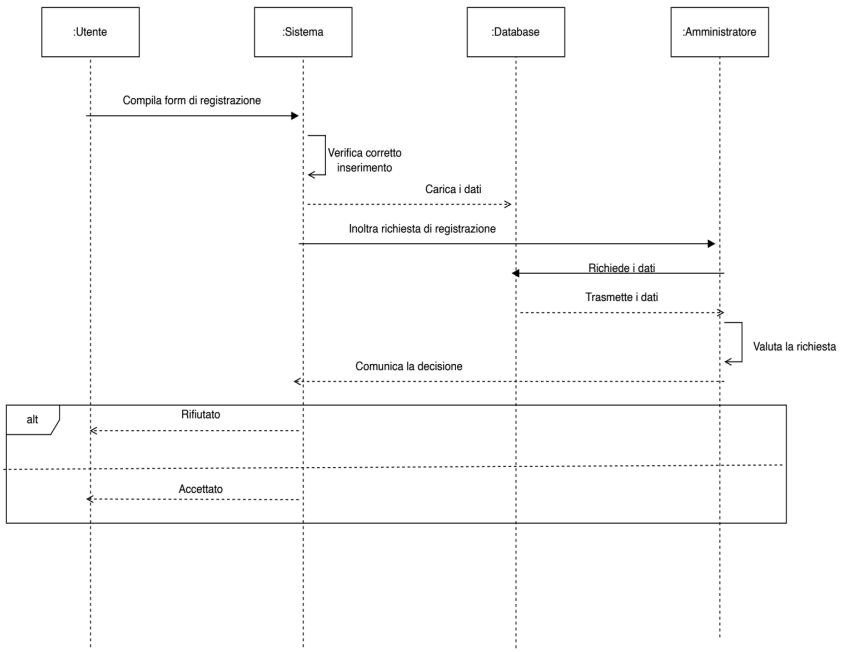
Come abbiamo detto, lo Use Case non considera l'avanzamento del tempo.
È doveroso dunque introdurre un altro dei diagrammi più importanti definite nel linguaggio di UML.

Stiamo parlando del Sequence diagram:
I Sequence Diagram, sono diagrammi di comportamento che modellano le interazioni tra varie entità di un sistema.
Mostrano lo scambio di messaggi nel corso del tempo tra ogni elemento,con l'obiettivo di mostrare come un tipo di comportamento viene creato e come si evolve.

Di seguito vengono mostrate le tabelle di specifica più interessanti dello Use Case, con focus di alcune mostrando il relativo sequence diagram.

2.1.1 Relativo agli Amministratori

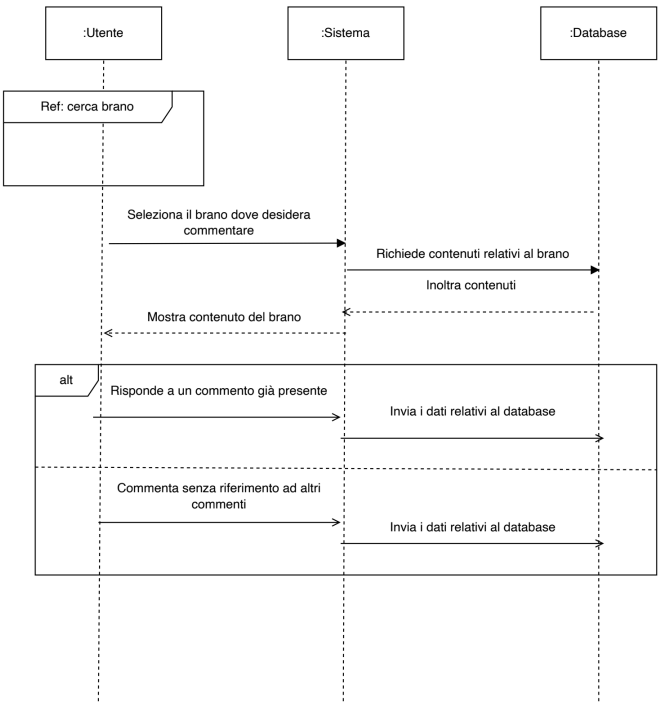
CASO D'USO UC-01	**Valuta registrazione utente**
ATTORI	Amministratore
PRECONDIZIONI	● login ● l'utente deve aver richiesto la registrazione
DESCRIZIONE	Decide di accettare o di rifiutare la richiesta di registrazione
POSTCONDIZIONI	Se l'utente viene approvato: ● potrà accedere al sistema tramite login Se l'utente non viene approvato: ● la richiesta viene scartata e l'utente non potrà accedere al sistema in nessun modo



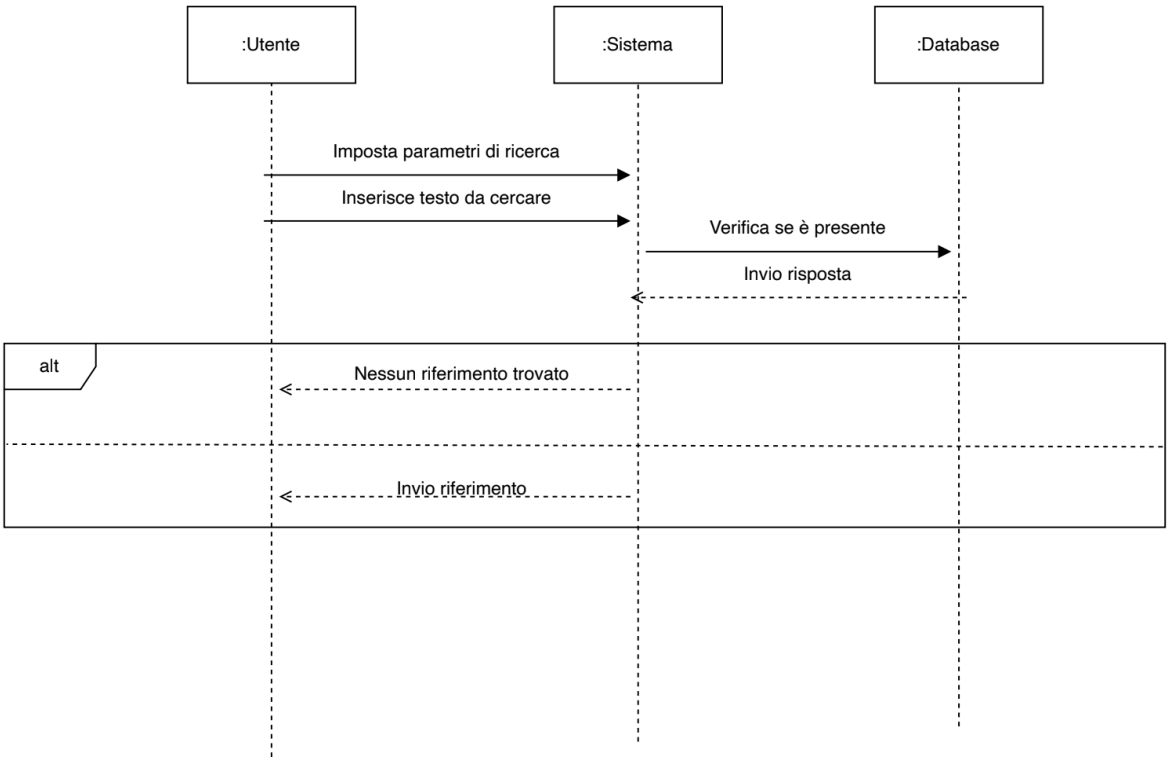
2.1.2 Relativo all' Utente

CASO D'USO UC-05	**Riproduce/scarica documento digitale**
ATTORI	Utente
PRECONDIZIONI	● login ● il documento digitale deve esistere
DESCRIZIONE	Ha la possibilità di: ● visualizzare/riprodurre in tempo reale il contenuto digitale ● scaricarlo sul proprio dispositivo scegliendo anche la posizione
POSTCONDIZIONI	Il documento viene riprodotto/scaricato sul dispositivo

CASO D'USO UC-06	**Commenta brano**
ATTORI	Utente
PRECONDIZIONI	● login ● il brano coinvolto deve esistere
DESCRIZIONE	Ha la possibilità di scrivere commenti relativi al brano, e di rispondere ad altri commenti. Se l'utente è autore/esecutore del brano, i suoi commenti verranno risaltati rispetto agli altri.
POSTCONDIZIONI	Il commento viene aggiunto e reso visibile a tutti gli utenti



CASO D'USO UC-10	**Ricerca brani/materiale relativo**
ATTORI	Utente
PRECONDIZIONI	● login ● esistono dati su cui operare
DESCRIZIONE	L'utente può effettuare una ricerca filtrata dei brani e del materiale relativo. La ricerca può essere per: - genere - titolo del brano - autore - esecutore/i
POSTCONDIZIONI	Se la ricerca ha avuto successo: ● l'utente visualizza i risultati che soddisfano la ricerca. Se la ricerca non ha avuto successo: ● non ci sono risultati che soddisfano la ricerca



2.1.3 Caso comune a entrambi gli attori

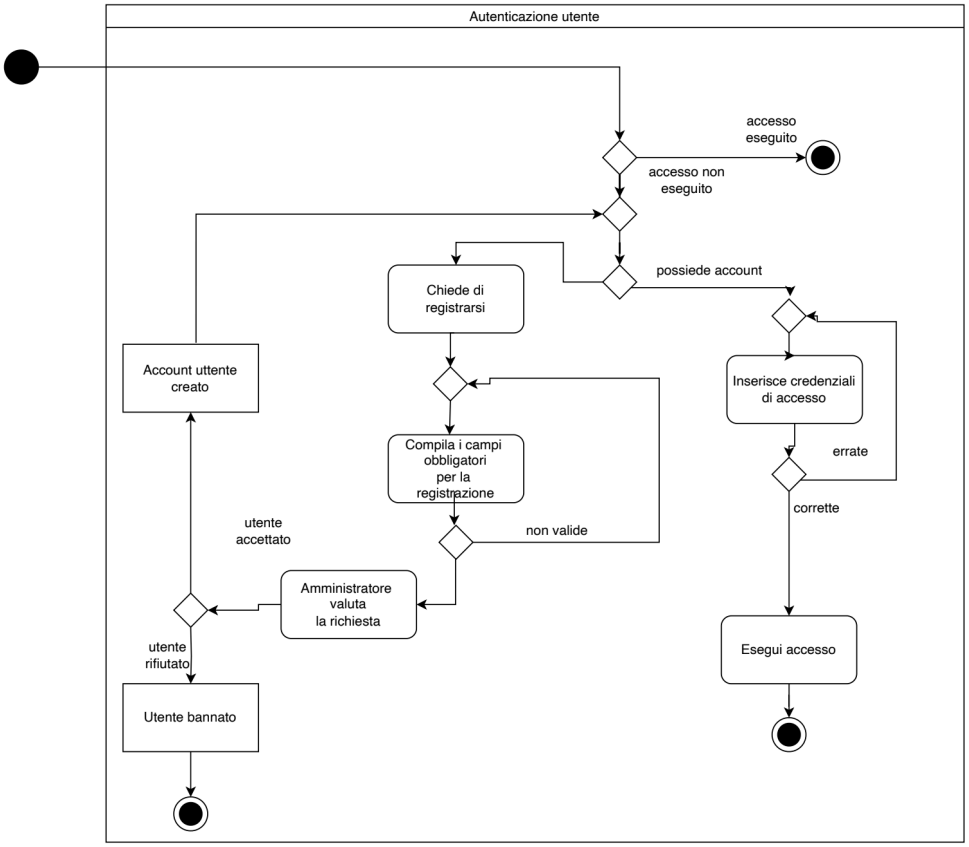
CASO D'USO UC-7	**Elimina commento**
ATTORI	Amministratore, utente
PRECONDIZIONI	● login ● bisogna avere i permessi per eliminare il commento
DESCRIZIONE	● l'amministratore può eliminare ogni commento ● l'utente ha il permesso di eliminare i commenti fatti da lui stesso e quelli fatti dagli altri in risposta al suo commento ● il creatore del brano può eliminare tutti i commenti relativi al brano
POSTCONDIZIONI	Nessuno visualizza più il commento eliminato

2.2 Activity diagram

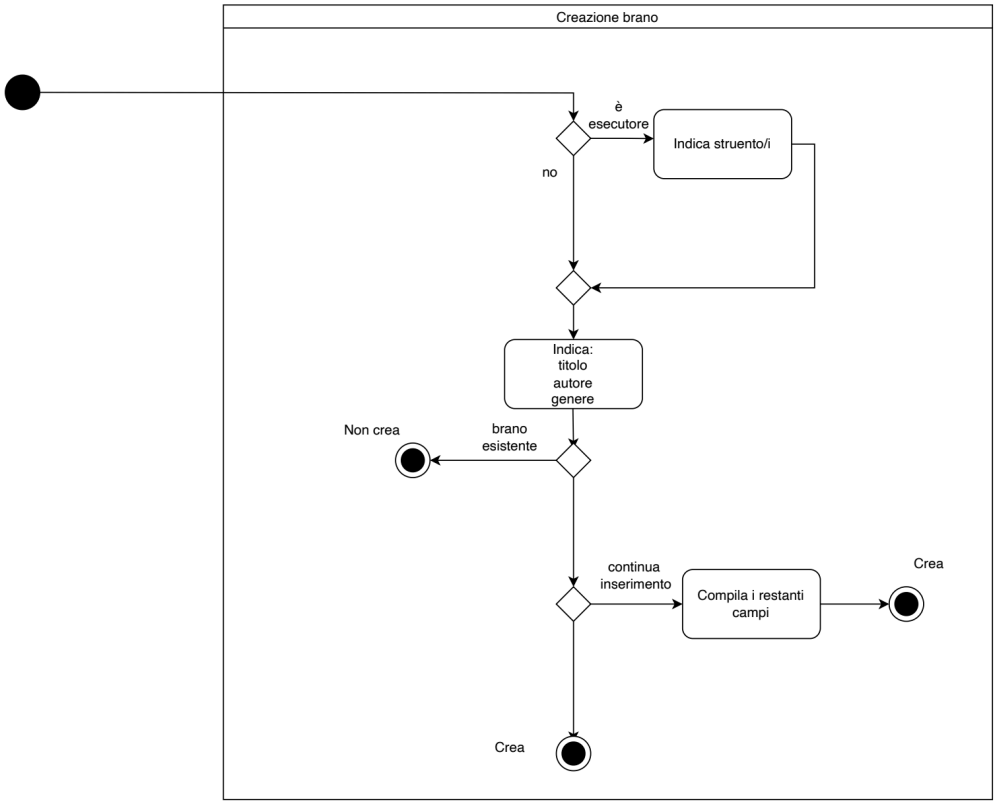
Un Activity Diagram è un diagramma utile a rappresentare il sistema e le varie interazioni al suo interno da un punto di vista differente.
Come i precedenti,anche in questo caso si tratta di una rappresentazione grafica comportamentale del sistema, però attraverso un flusso di controllo diverso.
Questo diagramma è utile per mettere in risalto le attività, le azioni e le decisioni che avvengono all'interno dell'esecuzione del sistema.

Di seguito vengono mostrati due diagrammi di attività fondamentali per questo sistema:

1. AUTENTICAZIONE DELL'UTENTE



2. UTENTE CREA UN BRANO



3 IMPLEMENTAZIONE SOFTWARE

3.1 Pattern architetturali e di design

I pattern architetturali sono soluzioni che permettono di organizzare la struttura complessa di un software.

Per questo software è stato utilizzato il pattern M-V-C (model, view, controller), integrato con il design pattern DAO (data access object). Quindi in questo caso i nostri modelli saranno dei semplici POJO.

Altri design pattern utilizzati sono stati:

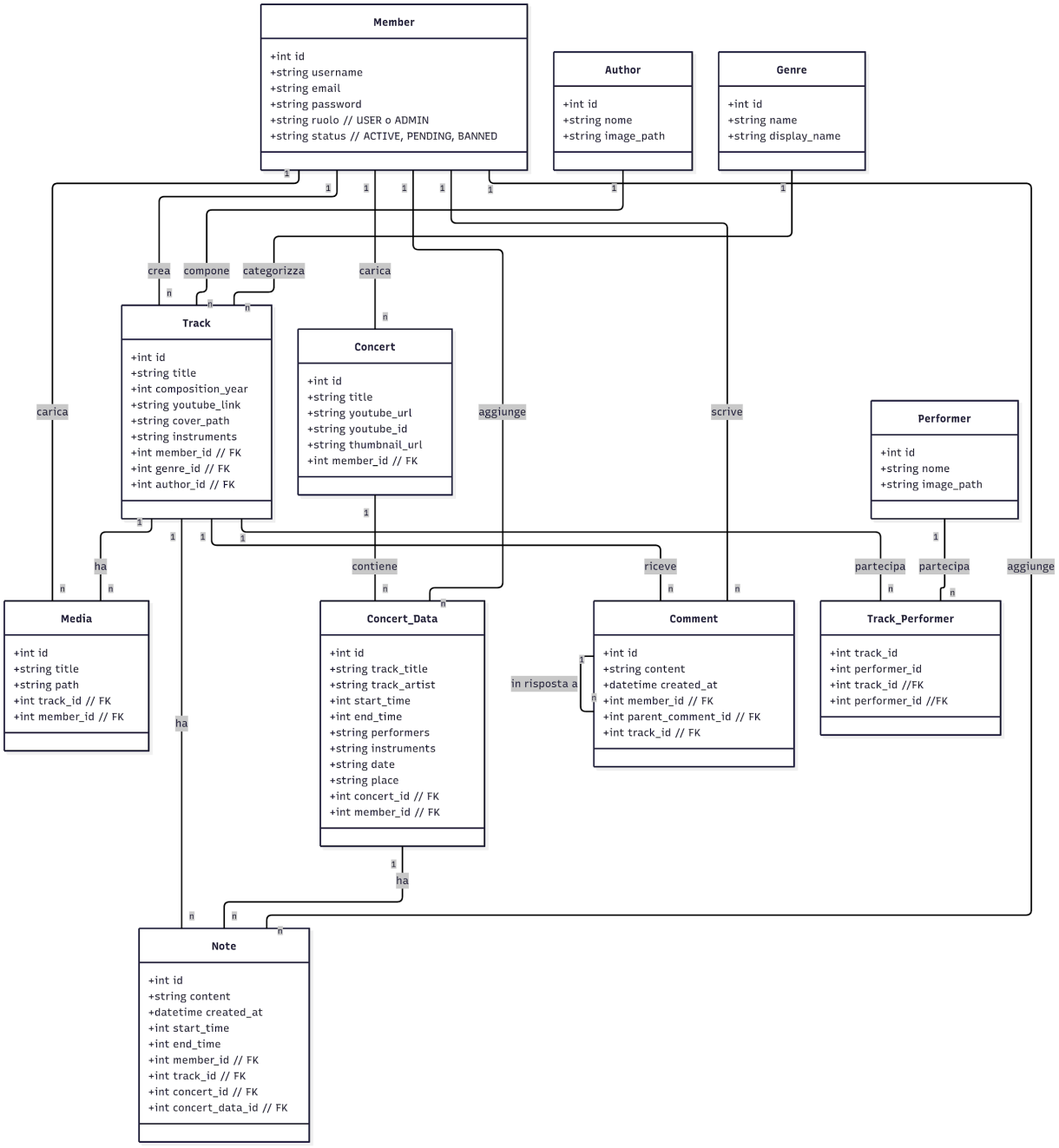
- Singleton
- Singleton variante lazy

3.2 Database diagram

Per il salvataggio dei dati inseriti nel sistema è stato utilizzato SQLite, un database relazionale embedded gestito tramite connessione JDBC (cioè che permette a un'applicazione java di collegarsi al database ed eseguire query SQL)

Di seguito viene mostrato il diagramma relativo alle tabelle che abbiamo creato.

Per gestire le relazioni molti a molti tra Track e Performer è stata creata una tabella di giunzione: "Track_Performer"



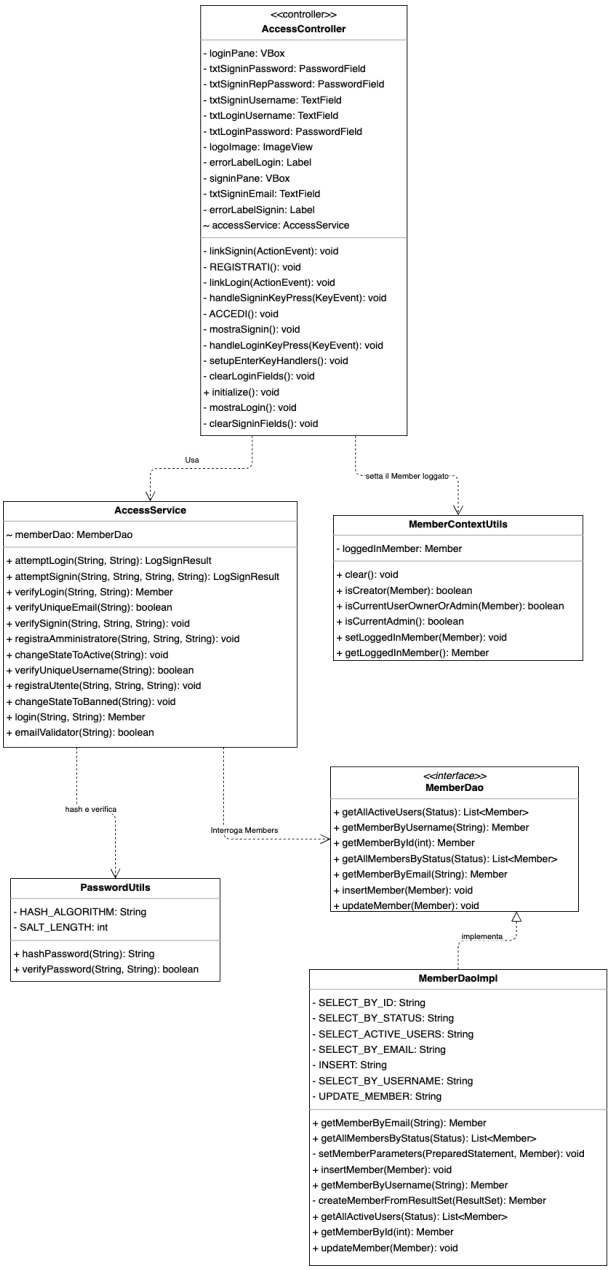
3.2 Class diagram

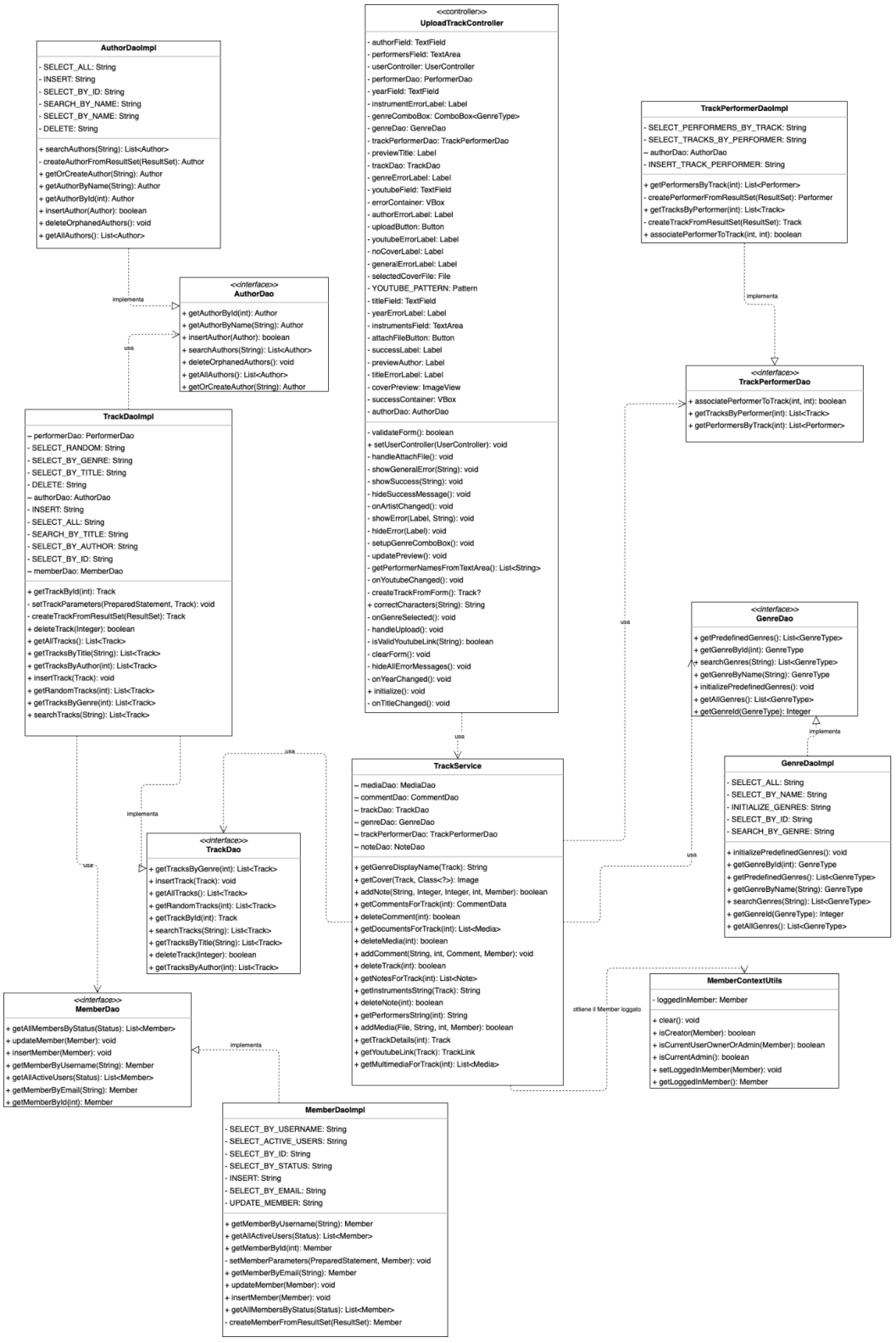
Un Class Diagram serve a rappresentare la struttura statica di un sistema, evidenziando le classi che lo compongono, i loro attributi, i metodi e le relazioni tra di essi.

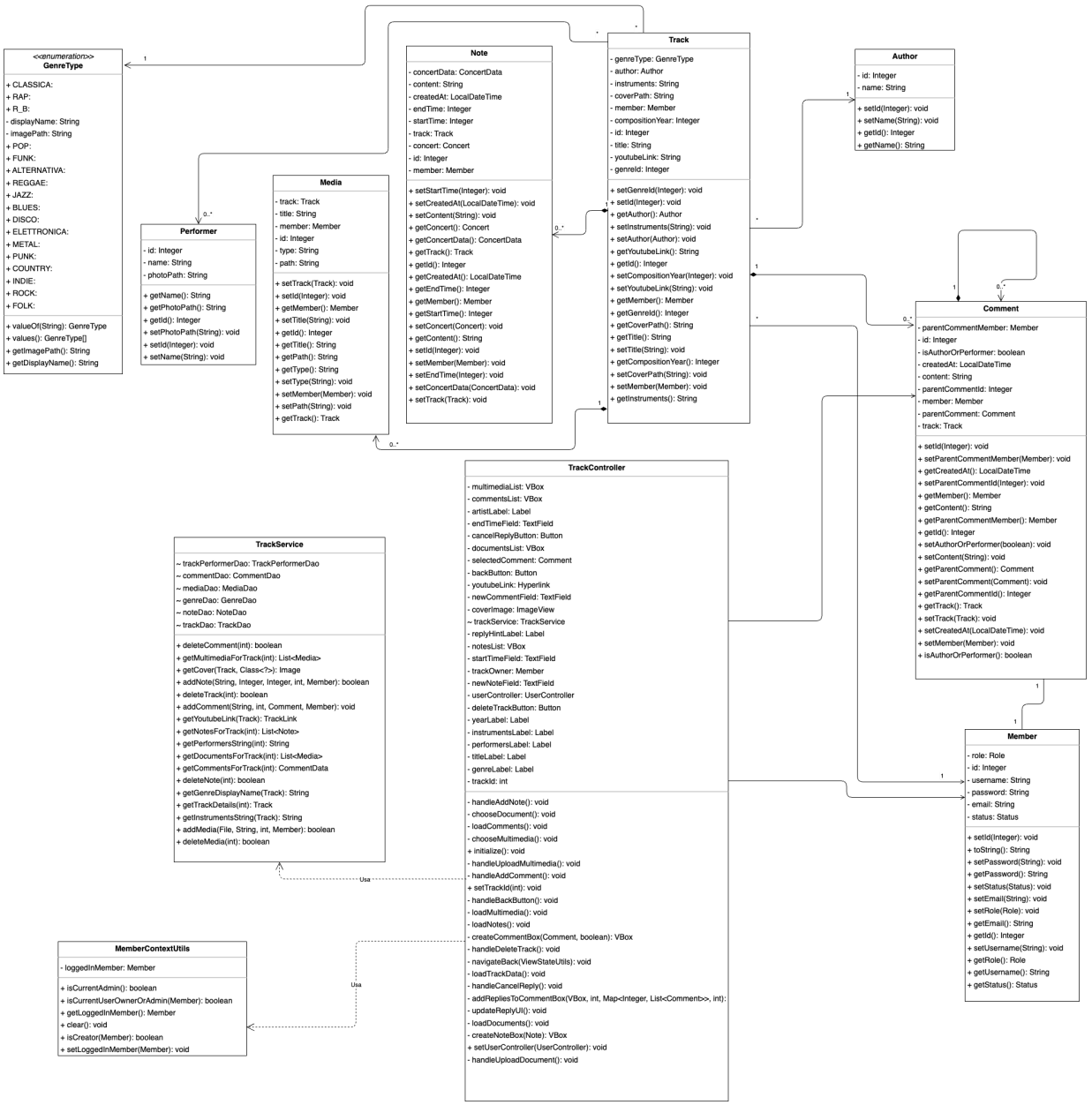
A differenza dei diagrammi comportamentali, come lo Use-Case e l'Activity Diagram, che si concentrano sul flusso delle attività o delle interazioni, il Class Diagram si focalizza sugli elementi strutturali del sistema.

La rappresentazione grafica permette di visualizzare le classi e le loro associazioni, mostrando anche le dipendenze tra classi.

Di seguito vengono mostrati i diagrammi relativi all' accesso, al caricamento del brano e al brano stesso.




AuthorDaoImpl
- SELECT_ALL: String
- INSERT: String
- SELECT_BY_ID: String
- SEARCH_BY_NAME: String
- SELECT_BY_NAME: String
- DELETE: String
+ searchAuthors(String): List<Author>
- createAuthorFromResultSet(ResultSet): Author
+ getOrCreateAuthor(String): Author
+ getAuthorByName(String): Author
+ getAuthorById(int): Author
+ insertAuthor(Author): boolean
+ deleteOrphanedAuthors(): void
+ getAllAuthors(): List<Author>
implementa
<<interface>>
AuthorDao
+ getAuthorById(int): Author
+ getAuthorByName(String): Author
+ insertAuthor(Author): boolean
+ searchAuthors(String): List<Author>
+ deleteOrphanedAuthors(): void
+ getAllAuthors(): List<Author>
+ getOrCreateAuthor(String): Author
usa
TrackDaoImpl
~ performerDao: PerformerDao
- SELECT_RANDOM: String
- SELECT_BY_GENRE: String
- SELECT_BY_TITLE: String
- DELETE: String
~ authorDao: AuthorDao
- INSERT: String
- SELECT_ALL: String
- SEARCH_BY_TITLE: String
- SELECT_BY_AUTHOR: String
- SELECT_BY_ID: String
~ memberDao: MemberDao
+ getTrackById(int): Track
- setTrackParameters(PreparedStatement, Track): void
- createTrackFromResultSet(ResultSet): Track
+ deleteTrack(Integer): boolean
+ getAllTracks(): List<Track>
+ getTracksByTitle(String): List<Track>
+ getTracksByAuthor(int): List<Track>
+ insertTrack(Track): void
+ getRandomTracks(int): List<Track>
+ getTracksByGenre(int): List<Track>
+ searchTracks(String): List<Track>
implementa
usa
<<interface>>
TrackDao
+ getTracksByGenre(int): List<Track>
+ insertTrack(Track): void
+ getAllTracks(): List<Track>
+ getRandomTracks(int): List<Track>
+ getTrackById(int): Track
+ searchTracks(String): List<Track>
+ getTracksByTitle(String): List<Track>
+ deleteTrack(Integer): boolean
+ getTracksByAuthor(int): List<Track>
usa
<<interface>>
MemberDao
+ getAllMembersByStatus(Status): List<Member>
+ updateMember(Member): void
+ insertMember(Member): void
+ getMemberByUsername(String): Member
+ getAllActiveUsers(Status): List<Member>
+ getMemberByEmail(String): Member
+ getMemberById(int): Member
implementa
MemberDaoImpl
- SELECT_BY_USERNAME: String
- SELECT_ACTIVE_USERS: String
- SELECT_BY_ID: String
- SELECT_BY_STATUS: String
- INSERT: String
- SELECT_BY_EMAIL: String
- UPDATE_MEMBER: String
+ getMemberByUsername(String): Member
+ getAllActiveUsers(Status): List<Member>
+ getMemberById(int): Member
- setMemberParameters(PreparedStatement, Member): void
+ getMemberByEmail(String): Member
+ updateMember(Member): void
+ insertMember(Member): void
+ getAllMembersByStatus(Status): List<Member>
- createMemberFromResultSet(ResultSet): Member
<<controller>>
UploadTrackController
- authorField: TextField
- performersField: TextArea
- userController: UserController
- performerDao: PerformerDao
- yearField: TextField
- instrumentErrorLabel: Label
- genreComboBox: ComboBox<GenreType>
- genreDao: GenreDao
- trackPerformerDao: TrackPerformerDao
- previewTitle: Label
- trackDao: TrackDao
- genreErrorLabel: Label
- youtubeField: TextField
- errorContainer: VBox
- authorErrorLabel: Label
- uploadButton: Button
- youtubeErrorLabel: Label
- noCoverLabel: Label
- generalErrorLabel: Label
- selectedCoverFile: File
- YOUTUBE_PATTERN: Pattern
- titleField: TextField
- yearErrorLabel: Label
- instrumentsField: TextArea
- attachFileButton: Button
- successLabel: Label
- previewAuthor: Label
- titleErrorLabel: Label
- coverPreview: ImageView
- successContainer: VBox
- authorDao: AuthorDao
- validateForm(): boolean
+ setUserController(UserController): void
- handleAttachFile(): void
- showGeneralError(String): void
- showSuccess(String): void
- hideSuccessMessage(): void
- onArtistChanged(): void
- showError(Label, String): void
- hideError(Label): void
- setupGenreComboBox(): void
- updatePreview(): void
- getPerformerNamesFromTextArea(): List<String>
- onYoutubeChanged(): void
- createTrackFromForm(): Track?
+ correctCharacters(String): String
- onGenreSelected(): void
- handleUpload(): void
- isValidYoutubeLink(String): boolean
- clearForm(): void
- hideAllErrorMessages(): void
- onYearChanged(): void
+ initialize(): void
- onTitleChanged(): void
usa
TrackService
~ mediaDao: MediaDao
~ commentDao: CommentDao
~ trackDao: TrackDao
~ genreDao: GenreDao
~ trackPerformerDao: TrackPerformerDao
~ noteDao: NoteDao
+ getGenreDisplayName(Track): String
+ getCover(Track, Class<?>): Image
+ addNote(String, Integer, Integer, int, Member): boolean
+ getCommentsForTrack(int): CommentData
+ deleteComment(int): boolean
+ getDocumentsForTrack(int): List<Media>
+ deleteMedia(int): boolean
+ addComment(String, int, Comment, Member): void
+ deleteTrack(int): boolean
+ getNotesForTrack(int): List<Note>
+ getInstrumentsString(Track): String
+ deleteNote(int): boolean
+ getPerformersString(int): String
+ addMedia(File, String, int, Member): boolean
+ getTrackDetails(int): Track
+ getYoutubeLink(Track): TrackLink
+ getMultimediaForTrack(int): List<Media>
ottiene il Member loggato
MemberContextUtils
- loggedInMember: Member
+ clear(): void
+ isCreator(Member): boolean
+ isCurrentUserOwnerOrAdmin(Member): boolean
+ isCurrentAdmin(): boolean
+ setLoggedInMember(Member): void
+ getLoggedInMember(): Member
TrackPerformerDaoImpl
- SELECT_PERFORMERS_BY_TRACK: String
- SELECT_TRACKS_BY_PERFORMER: String
~ authorDao: AuthorDao
- INSERT_TRACK_PERFORMER: String
+ getPerformersByTrack(int): List<Performer>
- createPerformerFromResultSet(ResultSet): Performer
+ getTracksByPerformer(int): List<Track>
- createTrackFromResultSet(ResultSet): Track
+ associatePerformerToTrack(int, int): boolean
implementa
<<interface>>
TrackPerformerDao
+ associatePerformerToTrack(int, int): boolean
+ getTracksByPerformer(int): List<Track>
+ getPerformersByTrack(int): List<Performer>
usa
<<interface>>
GenreDao
+ getPredefinedGenres(): List<GenreType>
+ getGenreById(int): GenreType
+ searchGenres(String): List<GenreType>
+ getGenreByName(String): GenreType
+ initializePredefinedGenres(): void
+ getAllGenres(): List<GenreType>
+ getGenreId(GenreType): Integer
implementa
GenreDaoImpl
- SELECT_ALL: String
- SELECT_BY_NAME: String
- INITIALIZE_GENRES: String
- SELECT_BY_ID: String
- SEARCH_BY_GENRE: String
+ initializePredefinedGenres(): void
+ getGenreById(int): GenreType
+ getPredefinedGenres(): List<GenreType>
+ getGenreByName(String): GenreType
+ searchGenres(String): List<GenreType>
+ getGenreId(GenreType): Integer
+ getAllGenres(): List<GenreType>
usa

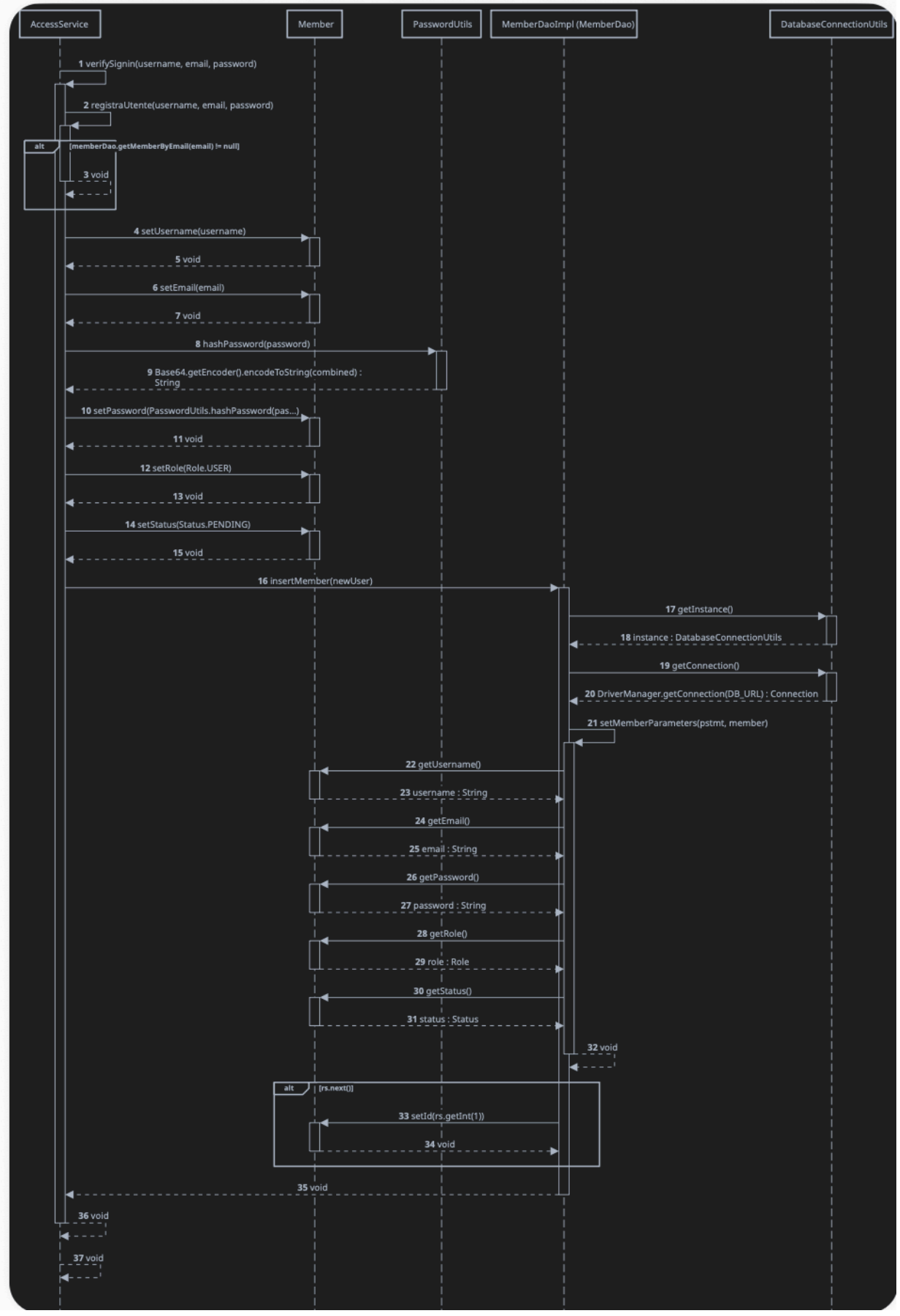

<<enumeration>>
GenreType
+ CLASSICA:
+ RAP:
+ R_B:
- displayName: String
- imagePath: String
+ POP:
+ FUNK:
+ ALTERNATIVA:
+ REGGAE:
+ JAZZ:
+ BLUES:
+ DISCO:
+ ELETTRONICA:
+ METAL:
+ PUNK:
+ COUNTRY:
+ INDIE:
+ ROCK:
+ FOLK:
+ valueOf(String): GenreType
+ values(): GenreType[]
+ getImagePath(): String
+ getDisplayName(): String
Performer
- id: Integer
- name: String
- photoPath: String
+ getName(): String
+ getPhotoPath(): String
+ getId(): Integer
+ setPhotoPath(String): void
+ setId(Integer): void
+ setName(String): void
Media
- track: Track
- title: String
- member: Member
- id: Integer
- type: String
- path: String
+ setTrack(Track): void
+ setId(Integer): void
+ getMember(): Member
+ setTitle(String): void
+ getId(): Integer
+ getTitle(): String
+ getPath(): String
+ getType(): String
+ setType(String): void
+ setMember(Member): void
+ setPath(String): void
+ getTrack(): Track
Note
- concertData: ConcertData
- content: String
- createdAt: LocalDateTime
- endTime: Integer
- startTime: Integer
- track: Track
- concert: Concert
- id: Integer
- member: Member
+ setStartTime(Integer): void
+ setCreatedAt(LocalDateTime): void
+ setContent(String): void
+ getConcert(): Concert
+ getConcertData(): ConcertData
+ getTrack(): Track
+ getId(): Integer
+ getCreatedAt(): LocalDateTime
+ getEndTime(): Integer
+ getMember(): Member
+ getStartTime(): Integer
+ setConcert(Concert): void
+ getContent(): String
+ setId(Integer): void
+ setMember(Member): void
+ setEndTime(Integer): void
+ setConcertData(ConcertData): void
+ setTrack(Track): void
Track
- genreType: GenreType
- author: Author
- instruments: String
- coverPath: String
- member: Member
- compositionYear: Integer
- id: Integer
- title: String
- youtubeLink: String
- genreId: Integer
+ setGenreId(Integer): void
+ setId(Integer): void
+ getAuthor(): Author
+ setInstruments(String): void
+ setAuthor(Author): void
+ getYoutubeLink(): String
+ getId(): Integer
+ setCompositionYear(Integer): void
+ setYoutubeLink(String): void
+ getMember(): Member
+ getGenreId(): Integer
+ getCoverPath(): String
+ getTitle(): String
+ setTitle(String): void
+ getCompositionYear(): Integer
+ setCoverPath(String): void
+ setMember(Member): void
+ getInstruments(): String
Author
- id: Integer
- name: String
+ setId(Integer): void
+ setName(String): void
+ getId(): Integer
+ getName(): String
Comment
- parentCommentMember: Member
- id: Integer
- isAuthorOrPerformer: boolean
- createdAt: LocalDateTime
- content: String
- parentCommentId: Integer
- member: Member
- parentComment: Comment
- track: Track
+ setId(Integer): void
+ setParentCommentMember(Member): void
+ getCreatedAt(): LocalDateTime
+ setParentCommentId(Integer): void
+ getMember(): Member
+ getContent(): String
+ getParentCommentMember(): Member
+ getId(): Integer
+ setAuthorOrPerformer(boolean): void
+ setContent(String): void
+ getParentComment(): Comment
+ setParentComment(Comment): void
+ getParentCommentId(): Integer
+ getTrack(): Track
+ setTrack(Track): void
+ setCreatedAt(LocalDateTime): void
+ setMember(Member): void
+ isAuthorOrPerformer(): boolean
Member
- role: Role
- id: Integer
- username: String
- password: String
- email: String
- status: Status
+ setId(Integer): void
+ toString(): String
+ setPassword(String): void
+ getPassword(): String
+ setStatus(Status): void
+ setEmail(String): void
+ setRole(Role): void
+ getEmail(): String
+ getId(): Integer
+ setUsername(String): void
+ getRole(): Role
+ getUsername(): String
+ getStatus(): Status
TrackController
- multimediaList: VBox
- commentsList: VBox
- artistLabel: Label
- endTimeField: TextField
- cancelReplyButton: Button
- documentsList: VBox
- selectedComment: Comment
- backButton: Button
- youtubeLink: Hyperlink
- newCommentField: TextField
- coverImage: ImageView
~ trackService: TrackService
- replyHintLabel: Label
- notesList: VBox
- startTimeField: TextField
- trackOwner: Member
- newNoteField: TextField
- userController: UserController
- deleteTrackButton: Button
- yearLabel: Label
- instrumentsLabel: Label
- performersLabel: Label
- titleLabel: Label
- genreLabel: Label
- trackId: int
- handleAddNote(): void
- chooseDocument(): void
- loadComments(): void
- chooseMultimedia(): void
+ initialize(): void
- handleUploadMultimedia(): void
- handleAddComment(): void
+ setTrackId(int): void
- handleBackButton(): void
- loadMultimedia(): void
- loadNotes(): void
- createCommentBox(Comment, boolean): VBox
- handleDeleteTrack(): void
- navigateBack(ViewStateUtils): void
- loadTrackData(): void
- handleCancelReply(): void
- addRepliesToCommentBox(VBox, int, Map<Integer, List<Comment>>, int): void
- updateReplyUI(): void
- loadDocuments(): void
- createNoteBox(Note): VBox
+ setUserController(UserController): void
- handleUploadDocument(): void
TrackService
~ trackPerformerDao: TrackPerformerDao
~ commentDao: CommentDao
~ mediaDao: MediaDao
~ genreDao: GenreDao
~ noteDao: NoteDao
~ trackDao: TrackDao
+ deleteComment(int): boolean
+ getMultimediaForTrack(int): List<Media>
+ getCover(Track, Class<?>): Image
+ addNote(String, Integer, Integer, int, Member): boolean
+ deleteTrack(int): boolean
+ addComment(String, int, Comment, Member): void
+ getYoutubeLink(Track): TrackLink
+ getNotesForTrack(int): List<Note>
+ getPerformersString(int): String
+ getDocumentsForTrack(int): List<Media>
+ getCommentsForTrack(int): CommentData
+ deleteNote(int): boolean
+ getGenreDisplayName(Track): String
+ getTrackDetails(int): Track
+ getInstrumentsString(Track): String
+ addMedia(File, String, int, Member): boolean
+ deleteMedia(int): boolean
MemberContextUtils
- loggedInMember: Member
+ isCurrentAdmin(): boolean
+ isCurrentUserOwnerOrAdmin(Member): boolean
+ getLoggedInMember(): Member
+ clear(): void
+ isCreator(Member): boolean
+ setLoggedInMember(Member): void
Usa
Usa

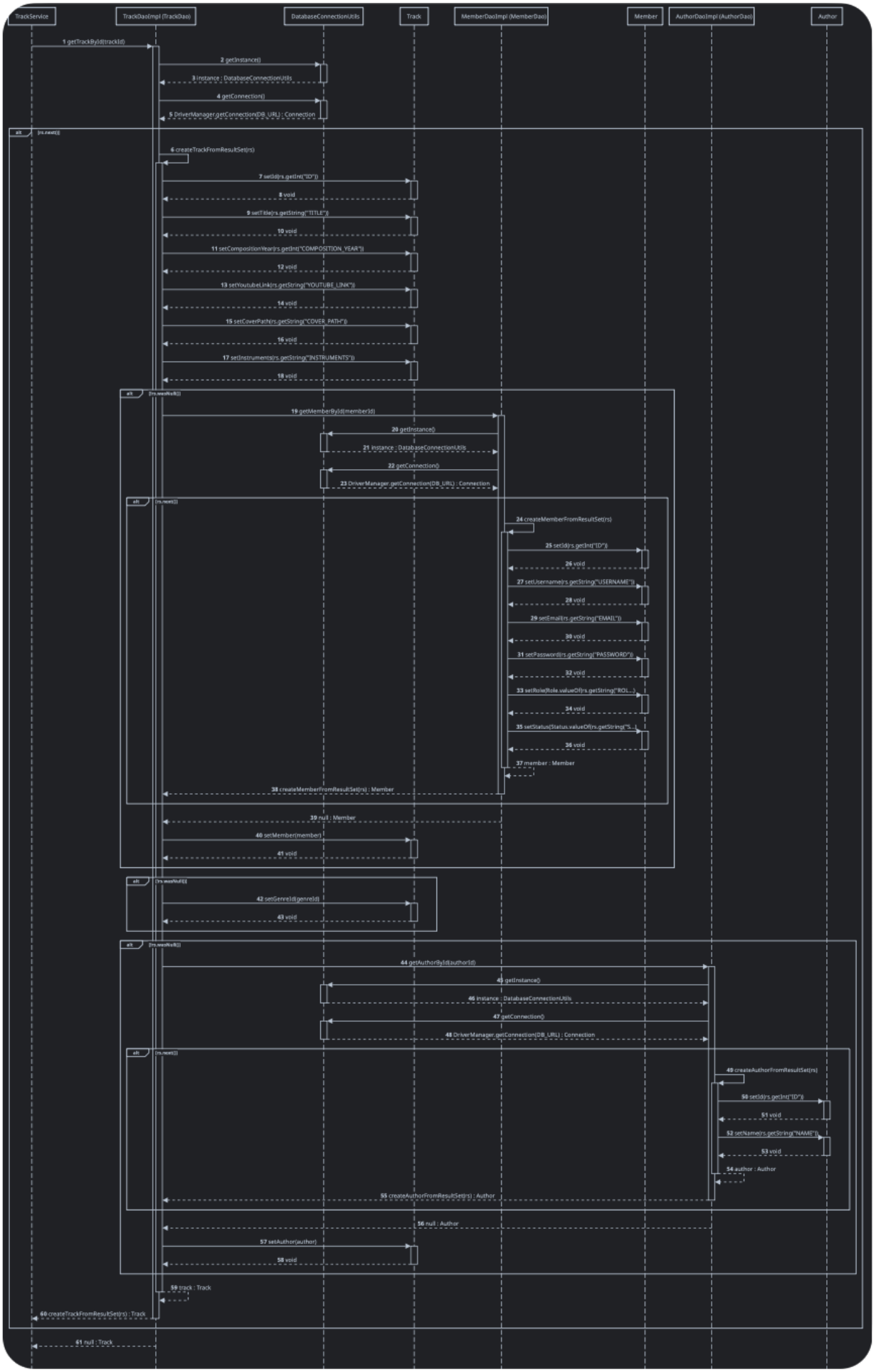
3.4 Sequence diagram a software completato

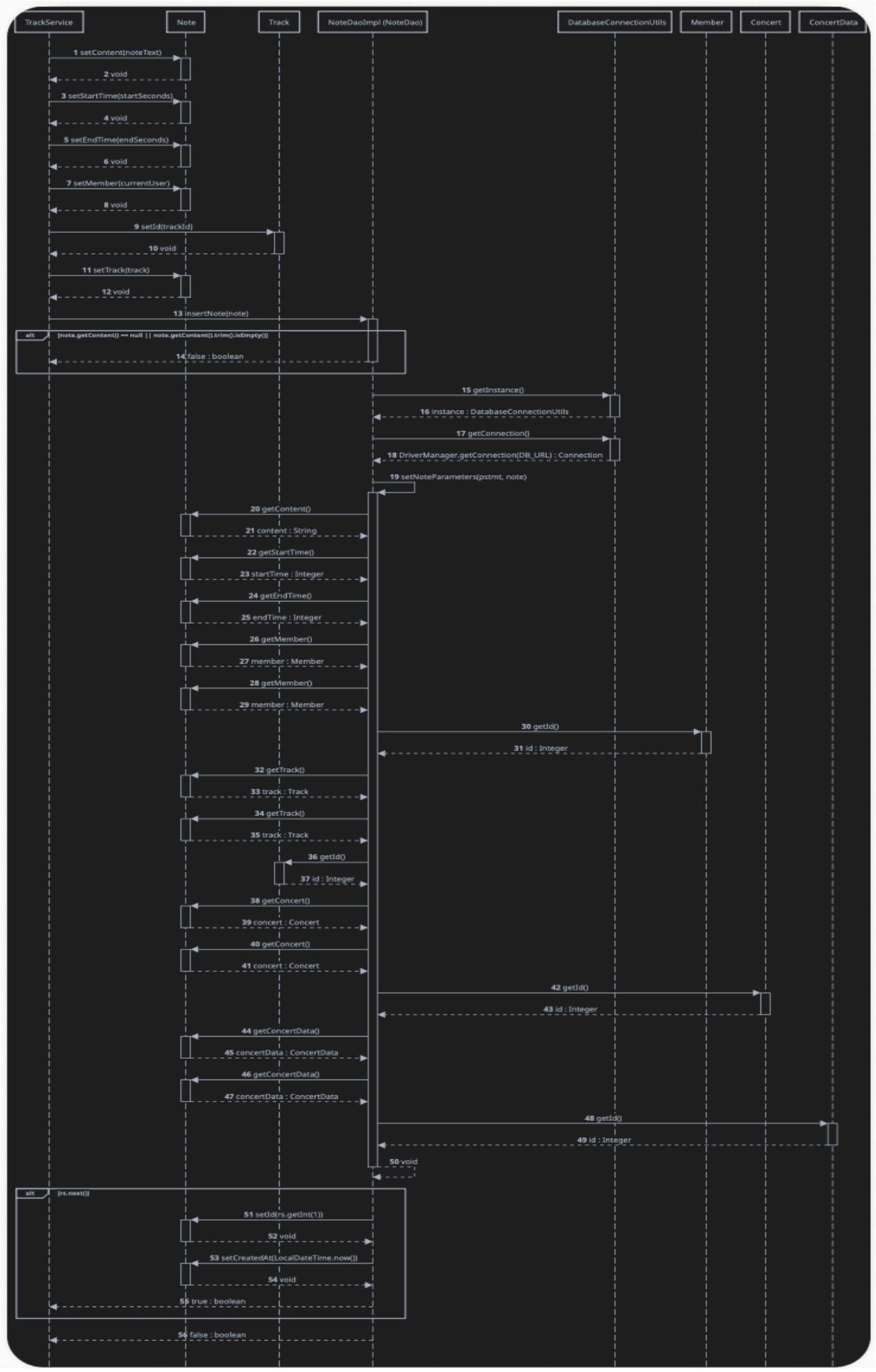
Di seguito vengono riportati i diagrammi di sequenza di alcune situazioni complesse, mettendo inoltre focus su ciò che non abbiamo avuto modo di vedere fino ad ora.

I diagrammi verranno rappresentati nel seguente ordine:

- ACCESSO AL SISTEMA
- VISUALIZZAZIONE BRANO
- INSERIMENTO NOTE ALL'INTERNO DEL BRANO


AccessService
Member
PasswordUtils
MemberDaoImpl (MemberDao)
DatabaseConnectionUtils
1 verifySignin(username, email, password)
2 registraUtente(username, email, password)
alt
[memberDao.getMemberByEmail(email) != null]
3 void
4 setUsername(username)
5 void
6 setEmail(email)
7 void
8 hashPassword(password)
9 Base64.getEncoder().encodeToString(combined) : String
10 setPassword(PasswordUtils.hashPassword(pas...)
11 void
12 setRole(Role.USER)
13 void
14 setStatus(Status.PENDING)
15 void
16 insertMember(newUser)
17 getInstance()
18 instance : DatabaseConnectionUtils
19 getConnection()
20 DriverManager.getConnection(DB_URL) : Connection
21 setMemberParameters(pstmt, member)
22 getUsername()
23 username : String
24 getEmail()
25 email : String
26 getPassword()
27 password : String
28 getRole()
29 role : Role
30 getStatus()
31 status : Status
32 void
alt
[rs.next()]
33 setId(rs.getInt(1))
34 void
35 void
36 void
37 void


TrackService
TrackDaoImpl (TrackDao)
DatabaseConnectionUtils
Track
MemberDaoImpl (MemberDao)
Member
AuthorDaoImpl (AuthorDao)
Author
1 getTrackById(trackId)
2 getInstance()
3 instance : DatabaseConnectionUtils
4 getConnection()
5 DriverManager.getConnection(DB_URL) : Connection
alt [rs.next()]
6 createTrackFromResultSet(rs)
7 setId(rs.getInt("ID"))
8 void
9 setTitle(rs.getString("TITLE"))
10 void
11 setCompositionYear(rs.getInt("COMPOSITION_YEAR"))
12 void
13 setYoutubeLink(rs.getString("YOUTUBE_LINK"))
14 void
15 setCoverPath(rs.getString("COVER_PATH"))
16 void
17 setInstruments(rs.getString("INSTRUMENTS"))
18 void
alt [rs.wasNotNull()]
19 getMemberById(memberId)
20 getInstance()
21 instance : DatabaseConnectionUtils
22 getConnection()
23 DriverManager.getConnection(DB_URL) : Connection
alt [rs.next()]
24 createMemberFromResultSet(rs)
25 setId(rs.getInt("ID"))
26 void
27 setUsername(rs.getString("USERNAME"))
28 void
29 setEmail(rs.getString("EMAIL"))
30 void
31 setPassword(rs.getString("PASSWORD"))
32 void
33 setRole(Role.valueOf(rs.getString("ROL...))
34 void
35 setStatus(Status.valueOf(rs.getString("S...))
36 void
37 member : Member
38 createMemberFromResultSet(rs) : Member
39 null : Member
40 setMember(member)
41 void
alt [rs.wasNull()]
42 setGenreId(genreId)
43 void
alt [rs.wasNotNull()]
44 getAuthorById(authorId)
45 getInstance()
46 instance : DatabaseConnectionUtils
47 getConnection()
48 DriverManager.getConnection(DB_URL) : Connection
alt [rs.next()]
49 createAuthorFromResultSet(rs)
50 setId(rs.getInt("ID"))
51 void
52 setName(rs.getString("NAME"))
53 void
54 author : Author
55 createAuthorFromResultSet(rs) : Author
56 null : Author
57 setAuthor(author)
58 void
59 track : Track
60 createTrackFromResultSet(rs) : Track
61 null : Track


TrackService
Note
Track
NoteDaoImpl (NoteDao)
DatabaseConnectionUtils
Member
Concert
ConcertData
1 setContent(noteText)
2 void
3 setStartTime(startSeconds)
4 void
5 setEndTime(endSeconds)
6 void
7 setMember(currentUser)
8 void
9 setId(trackId)
10 void
11 setTrack(track)
12 void
13 insertNote(note)
alt
[note.getContent() == null || note.getContent().trim().isEmpty()]
14 false : boolean
15 getInstance()
16 instance : DatabaseConnectionUtils
17 getConnection()
18 DriverManager.getConnection(DB_URL) : Connection
19 setNoteParameters(pstmt, note)
20 getContent()
21 content : String
22 getStartTime()
23 startTime : Integer
24 getEndTime()
25 endTime : Integer
26 getMember()
27 member : Member
28 getMember()
29 member : Member
30 getId()
31 id : Integer
32 getTrack()
33 track : Track
34 getTrack()
35 track : Track
36 getId()
37 id : Integer
38 getConcert()
39 concert : Concert
40 getConcert()
41 concert : Concert
42 getId()
43 id : Integer
44 getConcertData()
45 concertData : ConcertData
46 getConcertData()
47 concertData : ConcertData
48 getId()
49 id : Integer
50 void
alt
[rs.next()]
51 setId(rs.getInt(1))
52 void
53 setCreatedAt(LocalDateTime.now())
54 void
55 true : boolean
56 false : boolean

4 TEST

Nelle fasi finali della progettazione per verificare l'integrità del software prodotto sono state svolte diverse attività di test:

- Test degli sviluppatori sul sistema
- Unit Test su unità di codice
- Integration Test sulle funzionalità più critiche
- Test di utenti generici sul sistema.

4.1 Test degli sviluppatori

In questa fase di test gli sviluppatori hanno condotto in parallelo una serie di test funzionali per verificare il corretto funzionamento del software e identificare potenziali anomalie. Di seguito sono riportati i moduli testati e i relativi esiti.

- Modulo di Autenticazione e Autorizzazione: sono stati verificati i processi di registrazione e accesso, testando scenari con credenziali di utenti non approvati, sospesi, amministratori e utenti regolari. Il sistema ha dimostrato un comportamento prevedibile, negando l'accesso a utenti non autorizzati e gestendo correttamente la transizione tra i vari stati del profilo utente.
- Modulo di Inserimento Brano: è stata testata la raggiungibilità e la funzionalità del form, con particolare attenzione alla validazione dei dati in input. Tentativi di inserimento con campi obbligatori vuoti, parzialmente compilati o con formati non validi (ad esempio, caratteri testuali nel campo "anno") sono stati correttamente respinti, con il sistema che ha richiesto la correzione degli input.
- Modulo di Gestione Documenti: sono stati validati i controlli sul tipo di file, confermando che la sezione "Documenti" accetta esclusivamente file in formato PDF, mentre la sezione "Multimedia" consente solo l'upload di file MP3 e MP4. Il sistema impedisce efficacemente il caricamento di formati non supportati.
- Modulo di Inserimento Note e Commenti: il test ha verificato il comportamento del sistema in scenari limite, come l'inserimento di note vuote, contenenti solo un riferimento orario, o commenti privi di contenuto. In tutti i casi, le reazioni del sistema sono risultate conformi alle specifiche progettuali.
- Gestione dei Permessi di Eliminazione: è stato confermato che i pulsanti per l'eliminazione di contenuti sono visualizzati esclusivamente agli utenti con i privilegi appropriati, impedendo azioni non autorizzate.
- Modulo di Inserimento Concerto: il test si è concentrato sulla validazione dell'input, accertando che il form respinga input vuoti o stringhe non riconducibili a un link YouTube valido. Per i link validi, l'estrazione automatica dei metadati dal video avviene con successo.
- Modulo di Inserimento Metadati di Concerto: Sono stati verificati i controlli sull'input e l'integrità dei dati, incluso l'inserimento di brani, la prevenzione di sovrapposizioni temporali e il rispetto di una durata minima. L'aggiunta di metadati aggiuntivi non genera comportamenti anomali nel sistema.
- Motore di Ricerca: il test ha validato l'algoritmo di ricerca per somiglianza, verificando che ricerche di termini non presenti nel database (brani, generi, esecutori, autori) restituiscano risultati pertinenti e coerenti, senza generare errori.

Una volta accertati che questi test hanno restituito tutti esiti positivi e rispecchiano le specifiche richieste, gli sviluppatori sono passati a svolgere dei test automatici per cercare di rilevare problemi più profondi.

4.2 Unit Testing

Gli Unit Test sono test automatici che verificano il corretto funzionamento di singole unità di codice, tipicamente metodi o classi.

Lo scopo è assicurare che ogni unità operi come previsto.

Nel nostro caso specifico sono stati svolti 2 Unit Test sui Data Access Object più grandi e con logica più articolata:

In TrackDaoTest ci focalizziamo sulla gestione dei brani musicali (Track):

- vengono verificati i metodi getter e setter di Track, viene verificata la corretta manipolazione delle proprietà del brano e le relazioni con Author e Member
- vengono validati i dati necessari all'inserimento di un nuovo brano nel database, con check nei campi obbligatori e un range realistico per l'anno di composizione
- viene testata la logica della ricerca per titolo, autore e genere del brano, con il controllo dei parametri di ricerca e la simulazione dell'esecuzione.

I test simulano l'assegnazione dell'ID e non interagiscono con un database reale ma con uno che isola il contesto per le prove.

Tutto le prove effettuate hanno avuto successo.

In MemberDaoTest ci focalizziamo sulla gestione degli utenti (Member):

- vengono testati i metodi getter e setter di Member
- viene validato l'inserimento di un nuovo utente, incluso il corretto settaggio di campi obbligatori come username, email, ruolo e stato
- viene testata la logica di ricerca per email e username degli utenti
- viene testata la corretta gestione del cambio di stato utente (PENDING, ACTIVE, BANNED)
- viene verificata la gestione dei ruoli (ADMIN, USER)

Come per TrackDaoTest, i test operano su database che isola il contesto.

I test unitari svolti sia per la gestione dei brani che per la gestione degli utenti hanno garantito la correttezza delle operazioni fondamentali.

I risultati mostrano un'implementazione robusta e funzionante delle funzionalità principali di gestione dei dati.

4.3 Integration Testing

Gli Integration Test verificano il funzionamento corretto di più componenti software integrati insieme, al contrario dei Unit Test che isolano unità singole.

L'obiettivo di questi test è assicurare che le interazioni tra moduli, servizi e database funzionino come progettato.

Nel nostro caso specifico sono stati svolti 2 Integration Test sulle interazioni che sono risultate più complicate da implementare:

In TrackDocumentIntegrationTest vengono svolti test di interazione relativi alla gestione dei brani musicali e dei documenti ad esso associati:

- verifica l'integrazione del servizio TrackService con gli oggetti Track, Author , Member e Media
- controlla la validazione sulle proprietà del brano
- testa le relazioni tra Track, Author e Member per assicurare coerenza nei dati
- simula un inserimento completo di un brano, compreso il caricamento di media/file e l'accessibilità a questi.

Il test conferma che il TrackService e le componenti collegate e testate lavorano correttamente insieme.

In CommentSystemIntegrationTest vengono svolti test sul sistema di inserimento di commenti a vari livelli di profondità.:

- verifica la costruzione della struttura gerarchica dei commenti fino a 15 livelli di profondità
- integra il Dao dei commenti con un database simulato
- controlla le operazioni di inserimento, recupero e cancellazione a cascata dei commenti (eliminare un padre elimina i figli)
- testa la costruzione e la compatibilità dell'albero di gerarchia con le classi CommentData
- valuta le performance nell'elaborazione e gestione di gerarchie complesse

Il test dimostra che il sistema gestisce correttamente tutti gli aspetti legati all'innestamento di commenti mantenendo il rapporto padre-figlio e tutti gli aspetti testati risultano funzionanti.

I test di integrazione condotti garantiscono che le componenti principali del sistema funzionino correttamente sia che si tratti di gestione di brani e media o del sistema gerarchico dei commenti.

I risultati quindi confermano che il software è affidabile anche in scenari complessi di interazione.

4.4 Test di utenti generici

Come ultima fase di test il software è stato sottoposto ad individui la cui conoscenza sul progetto era nulla e distaccati dal mondo di sviluppo software.

Per ragioni pratiche e per non dover interferire con il processo agli individui sono state fornite credenziali di utenti precedentemente autorizzati dall'amministratore.

In questa fase sono stati riscontrati alcuni problemi di carattere prevalentemente estetico che sono stati prontamente analizzati e risolti e sono stati raccolti feedback positivi sia sul funzionamento sia sulla stabilità del sistema.

5 CONSIDERAZIONI FINALI

Abbiamo scelto questa traccia perché l'idea di realizzarla era molto stimolante, ed ora a progetto concluso possiamo soltanto che confermare il grande interesse in merito.

Durante la progettazione del software e realizzazione degli schemi abbiamo lavorato in pair designing, mentre per quanto riguarda il codice abbiamo lavorato in due diverse modalità:

- per le task più delicate abbiamo lavorato in pair programming.
- per il resto del codice non critico sono stati fondamentali per noi Google Drive e GitHub, che ci hanno permesso di lavorare in modo coordinato.

In generale abbiamo riscontrato difficoltà in diversi punti, ma principalmente riguardo alcune funzionalità implementate a livello di codice.

La prima riguarda lo use case: abbiamo trovato difficoltà nella realizzazione di esso perchè è stato complesso astrarre la traccia al punto giusto, senza creare ripetizione /confusione.

Per quanto riguarda la programmazione: le difficoltà riguardano l'implementazione del bottone 'back' che permette di tornare indietro dal punto in cui ci si trova (che abbiamo risolto con push e pop della vista precedente definite dentro uno stack), la creazione di più esecutori che abbiamo risolto tramite una tabella di join tra la traccia e gli esecutori.

Queste difficoltà affrontate con successo, hanno ritardato i test e la creazione dei diagrammi in riferimento al software finito.

I nostri obiettivi sono comunque stati raggiunti, anche se con dispiacere siamo stati costretti a non implementare delle migliorie che avevamo intenzione di fare.

In conclusione,

è stato un lavoro complesso ma molto soddisfacente, che ci ha permesso di arricchire di molto le nostre conoscenze.