UNIVERSITÀ DI PADOVA  **FACOLTÀ DI INGEGNERIA**

TESI DI LAUREA

DEFINIZIONE DI UN MODELLO E SVILUPPO DI UN FRAMEWORK PER L'ARCHIVIAZIONE E IL SUPPORTO ALL'ANALISI MUSICOLOGICA DI DOCUMENTI SONORI STORICI

Laureando: Daniele Colanardi

Relatore: Prof. Sergio Canazza

Correlatore: Ing. Federico Altieri, Ing. Niccolò Pretto

Corso di Laurea Magistrale in Ingegneria Informatica

Padova, 21/02/2017

Anno Accademico 2016/17

Sommario

La tesi esamina la metodologia di digitalizzazione e conservazione di documenti sonori utilizzata presso il Centro di Sonologia Computazionale dell'Università di Padova, su cui si basa il *PsKit*, correntemente in uso. Dall'analisi del software esistente e dai colloqui sostenuti con gli utilizzatori vengono elaborati dei requisiti e, successivamente, poste le basi per lo sviluppo una nuova piattaforma tramite la progettazione dalla base di dati.

Viene inoltre affrontata la progettazione ed implementazione di un'interfaccia web per l'accesso e l'analisi di documenti sonori storici, il cui scopo è agevolare la fruizione e studio nella loro interezza, affiancando al segnale audio le principali informazioni contestuali raccolte durante il riversamento, come foto e video. Vengono inoltre discusse le tecnologie utilizzate per lo sviluppo, che si pongono come base per l'implementazione della nuova piattaforma. Nelle conclusioni vengono proposti alcuni possibili sviluppi futuri.

Ringraziamenti

Ai miei genitori, perché avete sempre creduto in me, fin dall'inizio: se ce l'ho fatta è anche merito vostro.
Agli amici, perchè siete semplicemente voi.
A Ylenia, per il sostegno, l'aiuto e per avermi sopportato fino alla fine.
A Federico e Niccolò, per l'aiuto indispensabile e l'infinita pazienza.
Al prof. Canazza, per la disponibilità e il supporto.
Al prof. Ferro, per gli utili consigli.

Indice

Elenco delle figure

Capitolo 1

Introduzione alle procedure di conservazione e archiviazione

1.1 Introduzione

Lo sviluppo e la continua evoluzione delle scienze informatiche negli ultimi anni ha reso palese il loro ruolo, sempre più fondamentale, in campi spesso percepiti come puramente "umanistici", come l'archeologia, la filologia, il restauro di opere d'arte. Tra le discipline in forte crescita spicca la *musical cultural heritage*, specialmente nelle attività di conservazione di documenti sonori che, secondo gli esperti del settore, "idealmente, dovrebbe coinvolgere una figura di programmatore specializzato." [4]

Secondo un *report* ufficiale dell'UNESCO [7], oltre la metà del patrimonio culturale musicale del mondo è a rischio e potrebbe sparire in futuro. Le cause sono multiple: digitalizzare e/o creare copie richiede un ingente quantità di fondi, oltre che di vaste conoscenze multidisciplinari. Molti archivi non possiedono i fondi necessari e mancano delle giuste figure professionali e, di conseguenza, non possono acquisire e utilizzare gli strumenti tecnologici e metodologici necessari.

Definire un protocollo di conservazione che risulti efficace si rivela così problematico e non immediato: il patrimonio di documenti sonori storici, infatti, è caratterizzato da una grandissima varietà di standard, tecniche, supporti e materiali, nonché di pratiche e metodologie applicate dagli autori in fase di registrazione. È inimmaginabile quindi poter racchiudere l'intero contenuto informativo di un documento in una semplice traccia sonora: per poterlo salvaguardare nella sua interezza è necessario raccogliere e catalogare ogni dettaglio contestualmente utile e non per forza legato al contenuto audio. Quindi un approccio sistematico e ingegneristico a questo problema si rivela utile, se non fondamentale. [3]

Non basta conservare in condizioni ottimali i supporti originali per assicurarne la conservazione nel tempo: i processi di degradazione chimico-fisica dei materiali sono inevitabili. L'unico modo per poter garantire la sopravvivenza di un documento nel tempo è rinunciare alla sua materialità, trasferendo il suo contenuto informativo (e, di conseguenza, culturale) da un *media* a un altro. Questo processo viene definito di *rimediazione* [3], e non è esente da svantaggi, essendo

soggetta a errori nella procedura e nelle operazioni di copia. Inoltre, queste attività avvengono a distanza di molti anni dalla registrazione originale, quando sono presenti tecnologie e convenzioni diverse, che, seppur ottenendo risultati tecnici migliori, potrebbero influire al punto tale da non rispecchiare più le idee e le intenzioni dell'artista. Si può quindi facilmente intuire come questo processo non assicuri la totale neutralità, portando a un problema di autenticità filologica della copia, che ne risulta minacciata.

Ogni documento sonoro è strettamente legato al supporto fisico su cui viene registrato e, quindi, al dispositivo che ne permette la riproduzione e che ne definisce l'esperienza di ascolto: questi dispositivi, proprio come i documenti sonori, sono destinati a una rapida obsolescenza ed è di primaria importanza preservare le loro funzionalità e le loro peculiarità.

Tutti questi fattori portano quindi alla necessità, per stabilire una buona strategia di conservazione, di andare oltre al semplice riversamento o conversione di un segnale, richiedendo la collaborazione tra professionisti del campo IT, esperti nello stoccaggio dei dati (sia analogici che digitali), archivisti e musicologi esperti [5].

1.2 Conservazione digitale dei documenti sonori storici

1.2.1 Conservazione Passiva

Per conservazione passiva si intende la prevenzione e il contrasto alle principali cause di deterioramento del supporto fisico originale.

Una modalità di conservazione passiva, detta *indiretta*, cerca di mantenere intatte le caratteristiche dei supporti tramite stoccaggio in ambienti controllati e opportune attenzioni nell'utilizzo. La conservazione passiva *diretta*, invece, interviene sul supporto per stabilizzarne le condizioni e rallentare i processi di deterioramento a cui è soggetto.

Le strategie impiegate, siano esse dirette o meno, vanno adattate alla natura dei supporti e alle diverse proprietà dei materiali che li compongono, che potrebbero reagire in maniera diversa alle sostanze e tecniche impiegate.

1.2.2 Conservazione Attiva

Il già citato processo di *rimediazione* esprime appieno il concetto di conservazione *attiva*: salvaguardare i documenti nel tempo, facendone delle copie su nuovi supporti, siano essi dello stesso tipo degli originali o meno.

L'uso di tecniche di digitalizzazione dei documenti per lo stoccaggio a lungo termine è stato al centro delle attenzioni degli esperti verso la fine del XX secolo [8, 12]: le tecnologie disponibili, infatti, pur promettendo un rapido progresso, non potevano garantire risultati qualitativamente accettabili per l'uso archivistico (in termini di formati, compressioni, frequenze di campionamento, ecc.), mentre quelle di *storaging* non fornivano sufficiente capienza e affidabilità.

Un primo passo avanti si ebbe quando Smith, nel 1999, considerò la digitalizzazione come strategia per la fruizione, ma ancora inadatta per l'archiviazione [14].

D'altro canto, era impensabile riuscire a garantire la piena integrità e funzionalità dei supporti originali e dei relativi macchinari nel corso degli anni, motivo per cui prevalse il concetto di "preservare il contenuto, non il supporto": la digitalizzazione si rivelò come il miglior strumento per questo scopo, che favorì l'abbandono dell'idea di "conservare l'originale" per favorire quella di "distribuire è conservare" [5].

1.2.3 Il ruolo dell'Ingegneria Informatica nella conservazione

Il secondo problema della conservazione di documenti sonori, dopo la rimediazione, è l'archiviazione. Solo negli ultimi anni si è capito che questi documenti necessitano di uno stoccaggio in ambienti controllati esattamente quanto quelli cartacei, come i libri. I metodi di archiviazione atti alla conservazione del supporto originale, però, hanno richiesto agli archivi ingenti investimenti tecnici e finanziari, dovuti anche alla necessità (non presente per i libri) di dover mantenere una vasta varietà di strumenti di riproduzione adatti. Pur possedendo i documenti e i relativi macchinari, infatti, gli archivi mancano del carico di conoscenze tecnologiche necessario per poter garantire la corretta fruizione delle opere, dovendosi rivolgere sempre di più a nuove figure specializzate [17].

La grande varietà di supporti, macchinari e tecniche, nonché la grande mole di informazioni e metadati in essi contenuti, richiede un approfondito studio del problema e una corretta ingegnerizzazione delle soluzioni per garantire una duratura ed efficiente conservazione delle opere. Ogni aspetto richiede, infatti, un approccio scientifico e metodologico per poter essere modellato al meglio, dal processo di riversamento, alle tecniche di *storage*, alle pratiche di fruizione e accesso alle informazioni, alla loro catalogazione ed indicizzazione.

In questo contesto il ruolo dell'ingegneria è quello di modellare un sistema che supporti e standardizzi le procedure di digitalizzazione e conservazione delle opere, evidenziando, inoltre, i punti critici e maggiormente ripetitivi del processo, la cui corretta caratterizzazione può confluire nello sviluppo di strumenti software che mitighino le probabilità di inserimento involontario di errori nei dati gestiti, garantendone la validità nel tempo.

Il compito degli ingegneri, quindi, deve risultare armonico con le richieste dettate dalla necessità di conservare il patrimonio culturale, oltre che tecnico; la collaborazione fra le diverse figure professionali coinvolte diventa così non solo consigliata, ma essenziale.

1.3 Metodi e strategie per la conservazione e il restauro

1.3.1 Le prime proposte di Storm

Nel 1980, William Storm [15] propose per primo una soluzione alla mancanza di standard per il processo di conservazione, individuando due "legitimate directions", due approcci al trasferimento dell'opera su nuovi supporti tali da permetterne una corretta archiviazione e conservazione nel tempo.

Tipo I: Il primo approccio viene definito dall'autore "as the perpetuation of the sound of an original recording as it was initially reproduced and heard by the people of the era" [15].

L'attività dell'archivista dunque non si sarebbe limitata alla raccolta dei supporti, ma avrebbe dovuto includere il mantenimento del loro contenuto informativo, tramite un processo di *re-recording* che preservasse anche il lato storico dell'opera. La copia del segnale audio avrebbe, quindi, dovuto essere il più possibile fedele al periodo storico proprio del documento, mantenendo i mezzi (e i relativi limiti) disponibili al tempo; il documento diventa così un testimone storico non solo culturale, ma anche tecnico, raccontando come fosse possibile fruirlo all'epoca della composizione. L'uso delle attrezzature originali si presta al meglio per perseguire lo scopo di mantenere "come il documento suonasse originariamente al pubblico".

Tipo II: Il secondo approccio espande quanto detto per il *Tipo I*, ponendosi l'ambizioso obiettivo di ricercare quello che Storm definisce "true sound of the artist" e il "live sound of original performers". Storm propone quindi l'utilizzo di strumenti e tecnologie diverse da quelle originali, purché portino in maniera affidabile, oggettiva e verificabile [15] al raggiungimento dello scopo, superando i limiti di una riproduzione storicamente fedele come quella proposta dal *Tipo I*.

1.3.2 "Save history, not rewrite it"

La guida commissionata dall'UNESCO espande quanto suggerito da Storm, proponendo l'approccio "save history, not rewrite it", influenzato anche dal lavoro di Schüller [13], il cui punto chiave è "analizzare tutti gli aspetti tecnici e artistici del documento originale come base per definire a quale obiettivo puntare con il *re-recording*" [2].

Secondo Schüller l'utilizzo di attrezzature originali è giustificata solo qualora si voglia riscostruire il suono autentico di una registrazione, esattamente come era possibile ascoltare al tempo [13]. Questa necessità viene definta, al contrario di quanto proposto da Storm, esotica, poco comune.

Viene consigliato quindi di definire una procedura per estrarre il segnale alla miglior qualità possibile, limitando al minino le elaborazioni audio. Il primo passo consiste quindi nell'analisi delle alterazioni del segnale, classificabili in due categorie [2]:

- *intenzionali*, cioè l'utilizzo da parte dell'artista di sistemi di equalizzazione, *noise reduction*, ecc.
- *non intenzionali*, divise a loro volta in due gruppi: dovute a difetti nelle tecniche di registrazione o causate da errori di utilizzo delle apparecchiature, come velocità errate o un'errato allineamento dei nastri magnetici.

Per ottenere un segnale di migliore qualità è possibile compensare queste alterazioni abbassando, allo stesso tempo, il livello di fedeltà storica della nuova copia. Vengono pertanto proposte tre strategie che bilanciano in maniera diversa questi due aspetti [2]:

Tipo A: Assimilabile al *Tipo I* descritto da Storm, prevede di mantenere quanto originariamente era possibile ascoltare all'epoca della registrazione.

Tipo B: Ammette la compensazione delle alterazioni intenzionali e la risoluzione di errori di utilizzo delle apparecchiature, oltre che utilizzare strumenti di riproduzione moderni per minimizzare l'introduzione di ulteriori distorsioni nella copia.

Tipo C: Ammette l'ulteriore rimozione di tutti gli artefatti e le imperfezioni indotte dalle tecniche di registrazione utilizzate originariamente.

La strategia *Tipo B* prevede che il tecnico intervenga sul segnale basandosi su informazioni non necessariamente estraibili dallo stesso, introducendo così un margine di interpretazione. Le informazioni necessarie, di carattere tecnico-scientifico, sono però spesso proprie delle attrezzature originali (si vedano le curve di equalizzazione, per esempio), o in qualche modo ricavabili sperimentalmente con un buon grado di precisione. Questo permette di garantire una certa oggettività del processo, che rappresenta un buon compromesso fra fedeltà storica e possibilità di ulteriori elaborazioni, venendo quindi fortemente consigliata.

La strategia *Tipo C*, invece, prevede lo sviluppo di elaborazioni apposite *"used to compensate for non-linear frequency response, caused by imperfect historical recording equipment and to eliminate rumble, needle noise, or tape hiss"* [13]. Operazioni di questo tipo richiedono le conoscenze di un esperto restauratore, che sappia agire opportunamente su ciascun documento in esame, e vanno quindi documentate approfonditamente.

1.3.3 I vantaggi della digitalizzazione

Il processo di digitalizzazione viene in larga parte riconosciuto, ai giorni nostri, come il miglior metodo per la salvaguardia delle opere dal decadimento dei loro supporti originali. I vantaggi di questo approccio, però, vanno oltre all'aspetto prettamente materiale, aprendo le porte a possibilità prima non attuabili, come descritto in [10].

Il materiale digitale, infatti, supera le limitazioni dei supporti fisici: la possibilità di renderlo accessibile in rete ne permetterebbe l'accesso a più persone, anche contemporaneamente, senza la necessità che queste si rechino obbligatoriamente dove risiede l'archivio. Questa "democratizzazione" dell'accesso si riflette così in una maggiore ricerca, sia scientifica che artistica, da parte di studiosi ed esperti in tutto il mondo.

La dematerializzazione delle informazioni, inoltre, pone rimedio al problema dell'usura, tipico dei supporti fisici e dei macchinari: il supporto originale, a fine processo, potrà essere riposto e conservato senza necessità di accederci nuovamente. Conseguentemente, vengono meno le complicate e delicate procedure di *setup* richieste per la riproduzione, sostituite dall'uso di un comune calcolatore.

Il passaggio al dominio digitale permette così l'accesso non solo a persone fisiche, ma anche a software con varie finalità, dalla ricerca, all'elaborazione, all'accesso (come già accennato), al supporto all'archiviazione.

Gli strumenti software discussi nei prossimi capitoli sono un esempio chiaro di come la digitalizzazione può espandere le possibilità offerte alle entità che curano la conservazione delle opere, così come alle attività di studio e di analisi.

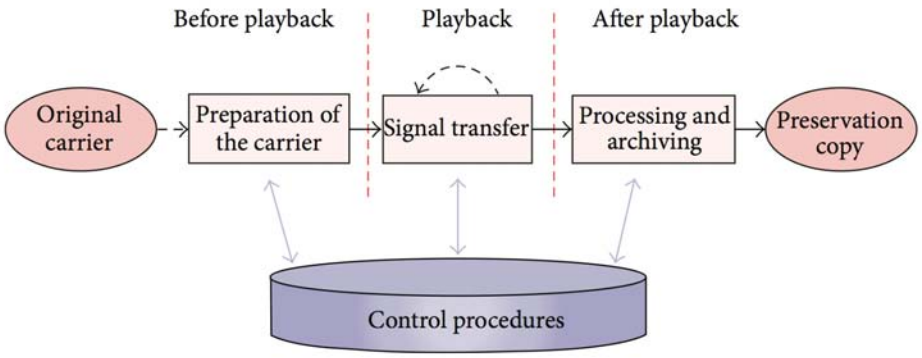


Figura 1.1: *Schema del processo di rimediazione proposto al CSC di Padova.*

1.4 Il processo di rimediazione al CSC

In [3] viene descritto il processo di rimediazione adottato presso il Centro di Sonologia Computazionale dell'Università degli Studi di Padova (CSC), un cui schema è riportato in Figura 1.1. Partendo da un documento sonoro, tramite degli step ben definiti di restauro e digitalizzazione, sarà possibile ottenere una copia conservativa digitale di tutto il contenuto informativo del documento, così come descritto in seguito.

1.4.1 Preparazione del supporto

Inizialmente l'operatore documenta lo stato dell'opera tramite foto e scansioni del supporto e di tutto il materiale che lo accompagna, come fogli, libretti e custodie.

Dopo aver validato la documentazione prodotta perchè sia utilizzabile, l'operatore ispeziona il supporto per rivelare le condizioni chimico-fisiche dello stesso e l'eventuale presenza di muffe, polvere o altri danni (vedi 1.5).

Infine agisce, tramite interventi mirati, alla risoluzione di queste problematiche, eliminando ogni forma di contaminazione. Il supporto viene infine preparato per essere caricato nel riproduttore, fornendo o sostituendo i componenti mancanti o danneggiati come, ad esempio, la flangia di una bobina.

1.4.2 Transferimento del segnale

Prima di procedere è richiesto che l'operatore conosca tutti i parametri di configurazione della macchina necessari per ottenere una corretta riproduzione del segnale contenuto nel supporto.

Questi parametri variano per ogni tipologia di supporto e, solitamente, sono documentati insieme a questo. Qualora queste informazioni mancassero, sarà carico dell'operatore determinare i parametri richiesti, mediante ricerche e prove sperimentali. Un'esperimento effettuato al CSC di Padova ha rivelato che determinare le giuste curve di equalizzazione a orecchio potrebbe contaminare la copia conservativa con considerazioni soggettive, ma allo stesso tempo anche figure non esperte possono raggiungere risultati validi seguendo un adeguato, ma breve, training [16]. I

parametri più frequenti sono, oltre alla già citata curva di equalizzazione, velocità (di scorrimento del nastro o di rotazione) e sistema di riduzione del rumore.

Dopo aver raccolto tutte queste informazioni, vengono preparati il riproduttore e tutti gli strumenti utili alla rimediazione: per il trasferimento in digitale saranno necessari infatti schede audio per l'acquisizione (*analog to digital converter*, o ADC), casse monitor per l'ascolto e PC su cui eseguire il software di registrazione ed elaborazione.

Il documento può essere quindi riprodotto: durante questa fase è necessaria la presenza e il monitoraggio costante da parte dell'operatore per poter documentare e intervenire per ogni problema che potrebbe insorgere. L'operatore stesso, durante il monitoraggio, dovrà documentare tutte le alterazioni riconoscibili del segnale audio, come rumori di fondo temporanei e globali (che perdurano per tutta la durata della riproduzione), rumori elettrici, o altre sue degradazioni.

Una volta completata l'acquisizione, l'operatore eseguirà una serie di validazioni, sia automatiche (tramite software) che manuali, che verifichino la qualità della copia appena creata. Nel caso in cui si palesino errori operativi o non imputabili al segnale originale andrà ripetuto il procedura di trasferimento, al fine di ottenere la miglior rappresentazione digitale del contenuto audio sorgente.

Quando la copia viene considerata valida, il supporto originale verrà rimosso dal riproduttore e preparato per essere riarchiviato, documentandone le condizioni a fine processo.

1.4.3 Elaborazione e archiviazione

Come ultimo passo vengono estratti tutti i metadati di foto, audio e video prodotti, riuniti in una struttura predefinita che possa contenere tutte le informazioni relative al documento e corredati di tutto il materiale raccolto durante il processo.

Si otterrà così una copia digitale del documento che racchiude l'intero contenuto informativo dello stesso, e che potrà essere archiviata in appositi sistemi di storage (ad es., un NAS).

Nel Capitolo 2 verrà descritta la composizione di queste copie conservative nel dettaglio.

1.5 Tipologie di supporti

Segue una panoramica dei più diffusi supporti audio e dei relativi problemi di deterioramento che li affliggono.

L'*International Association of Sound and Audiovisual Archives* (IASA) ha stilato un'approfondita analisi sui supporti, le loro composizioni chimico-fisiche e i principi di registrazione, nonché sulle tecniche di restauro e conservazione passiva da seguire per ottimizzare la conservazione dei documenti [1].

1.5.1 Supporti meccanici

I supporti meccanici sono accomunati dal metodo di registrazione: le informazioni vengono impresse, tramite appositi bracci incisori, con un solco sulla superficie modulato dal segnale sonoro. In questa categoria di supporti possiamo trovare: cilindri fonografici, dischi a macro e

(a) **(b)**

Figura 1.2: *Un cilindro fonografico (a) e un disco in vinile (b).*

micro solchi, sia stampati che registrabili. Periodo storico e composizione sono riportati nella Tabella 1.1.

Supporto	**Periodo**	**Composizione**
Cylinder	1886-1950s	Cera
Replicated cylinder	1902-1929	Cera e nitrocellulosa con plaster (blue amberol)
Coarse groove disc replicated	1887–1960	Polvere minerale su legante organico (shellac)
Coarse and microgroove discs recordable ("instantaneous discs")	1930–1950s	Lamina di Acetato o nitrato di cellulosa su alluminio, vetro, acciaio, carta
Microgroove disc (vinyl) replicated	1948-oggi	Polivinilacetato (PVCA)

Tabella 1.1: *Tipologie di supporto meccanici analogici*

Le principali cause di deterioramento che affliggono i supporti meccanici sono:

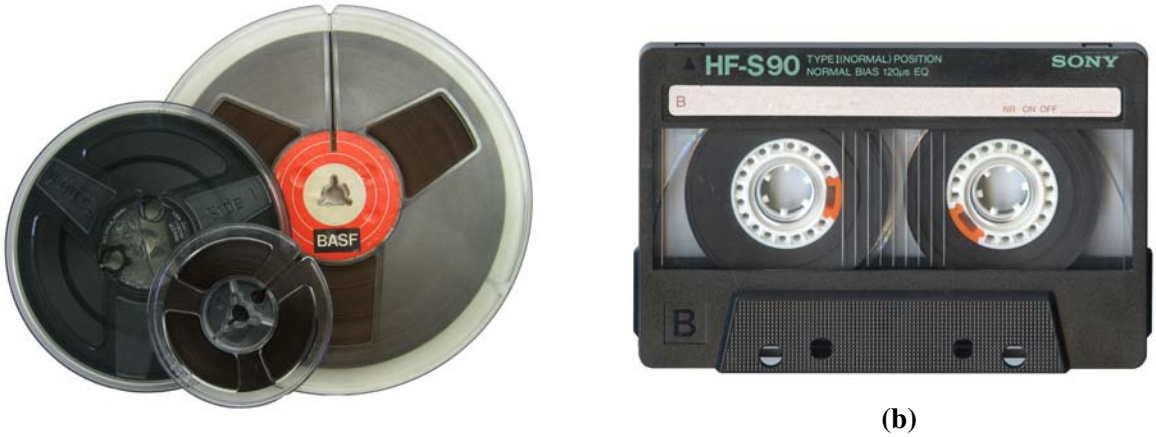
Umidità: I materiali che compongono questi supporti, a esclusione di gommalacca e vinile, sono soggetti a idrolisi, cioè reagiscono con le molecole d'acqua scindendosi in più parti. Inoltre, possono essere affetti dalla formazione di funghi, che avviene a una percentuale di umidità superiore al 65%. È facile notare quindi come questo fattore sia estremamente pericoloso per l'integrità dei supporti.

Temperatura: Materiali come vinile e cera sono estremamente suscettibili a temperature superiori ai 40°C, oltre i quali potrebbero deformarsi. In ogni caso, è consigliato mantenere una temperatura bassa, per rallentare le reazioni chimiche, e stabile, per evitare deformazioni da dilatazione termica.

Deformazioni meccaniche: Materiali come lo shellac sono particolarmente fragili, mentre altri, come il vinile, sono più soggetti a deformazioni plastiche. Graffi e deformazioni possono corrompere la struttura dei solchi che codificano l'informazione sonora, per cui è necessario trattare con estrema cura i supporti, e, tranne casi particolari, mantenere i dischi in posizione verticale.

Polvere e sporco: La presenza di particelle estranee nei solchi dei dischi può deviare la testina di lettura, introducendo fruscii, *click* e altri disturbi in fase di riproduzione. È necessario quindi operare in un ambiente libero da polvere ed evitare di lasciare impronte digitali sul supporto (ottimo adesivo per la materia estranea), utilizzando, ad esempio, dei guanti.

1.5.2 Nastri magnetici



(a) **(b)**

Figura 1.3: *Delle bobine (a) e una cassetta (b).*

La registrazione su nastri magnetici nasce come evoluzione di quella su filo magnetico sviluppata a fine '800; sviluppata in Germania negli anni '30, diventerà di uso comune solamente negli anni '50. Il segnale impiegato è tipicamente analogico, ma nel corso degli anni si sono sviluppati metodi di registrazione digitale. I nastri utilizzati sono solitamente raccolti in bobine o cassette. Nella Tabella 1.2 sono esposte le diverse tipologie di supporti.

Le principali cause di deterioramento che affliggono i nastri magnetici sono:

Umidità: Come descritto in 1.5.1, l'umidità favorisce processi di idrolisi e la crescita di funghi sulla superficie dei supporti, che può distruggere il pigmento magnetico o impedirne la lettura.

Substrato	**Pigmento magnetico**	**Tecnica di registrazione**	**Formati**	**Periodo**
Acetato di cellulosa	Fe_2O_3	Analogica	Bobine	1935-1960
PVC	Fe_2O_3	Analogica	Bobine	1944-1960
Poliestere	Fe_2O_3	Analogica	Bobine, Compact Cassette IEC I	1959-oggi
Poliestere	CrO_2	Analogica/Digitale	Compact Cassette IEC II, DCC	1969-oggi
Poliestere	SLH Ferro (blue tabs, Type I), Fe-Cr (red tabs, Type II)	Analogica	Elcaset	1976-1980
Poliestere	Particolato metallico	Analogica/Digitale	Compact Cassette IEC IV, R-DAT	1979-oggi

Tabella 1.2: *Tipologie di nastri magnetici*

Temperatura: Condizioni sfavorevoli o incontrollate di temperatura e umidità possono portare a dilatazioni e deformazioni dei supporti, oltre che favorire reazioni chimiche causa di degrado del materiale. Maggiore la densità delle informazioni contenute nel supporto, maggiore è la suscettibilità a variazioni dimensionali. I materiali esatti utilizzati per i pigmenti magnetici sono spesso posti a segreto industriale, per cui definire delle condizioni ideali richiede molta ricerca, affidandosi anche all'intervento di esperti in chimica.

Integrità meccanica: I nastri devono essere riavvolti senza lasciare alcun bordo esposto, poichè ne minerebbe la leggibilità e l'usabilità. Ogni deformazione del supporto deve essere evitata. Particolari e periodiche cure devono essere dedicate ai riproduttori, perchè non danneggino il nastro durante la lettura.

Polvere e sporco: La presenza di molecole estranee sulla superficie del nastro impedisce alla testina magnetica la lettura del segnale, soprattutto per i formati con elevata densità di informazione. Smog e fumo inoltre possono accelerare alcune reazioni e, quindi, il degrado del supporto.

Campi magnetici parassiti: Attrezzature come microfoni dinamici, coni dei monitor e altri componenti induttivi possono generare dei campi magnetici che interagiscono con il pigmento magnetico, intaccando le informazioni contenute al suo interno.

1.5.3 Dischi ottici

I dischi ottici digitali sfruttano una serie di avvallamenti in un materiale plastico posto al di sopra di uno strato riflettente. L'alternanza di questi variazioni di spessore modifica le capacità

(a)

(b)

Figura 1.4: *Fronte stampato (a) e retro riflettente (b) di un CD Audio.*

riflettenti del supporto, permettendo così di leggere lo stream di bit registrato. Questi supporto necessitano di essere conservati in apposite custodie per proteggere la superficie riflettente, completamente esposta.

Formato	**Substrato**	**Strato riflettente**	**Tecnica di registrazione**	**Periodo**
CD	Policarbonato	Alluminio, vernice, inchiostri	Digitale	1981-oggi
CD-R	Policarbonato	Oro, argento, vernice, inchiostri	Digitale	1992-oggi
CD-RW	Policarbonato	Vernice, inchiostri	Digitale	1996-oggi
SACD	Policarbonato	Alluminio	Digitale	1999-oggi
DVDA	Policarbonato	Alluminio (interno), oro (esterno semiriflettente)	2000-oggi	

Tabella 1.3: *Tipologie di dischi ottici digitali*

Le principali cause di deterioramento che affliggono i nastri magnetici sono:

Decolorazione: Lo strato riflettente, per invecchiamento o temperature non adatte, potrebbe ossidarsi o cambiare colore, intaccando le sue stesse proprietà riflettenti.

Muffa: Cattive condizioni di temperatura e umidità possono favorire la crescita di muffe, che spesso "assumono la forma di macchie bianche o grigiastre, con una struttura caratteristica visibile già a bassi ingrandimenti" [11].

Integrità meccanica: Al contrario dei supporto già affrontati, l'impossibilità di leggere correttamente anche solo pochi bit del contenuto potrebbe compromettere la leggibilità di intere sezioni, qualora l'algoritmo di correzione degli errori non sia in grado di compensare le lacune. In casi più gravi potrebbe non essere possibile leggere l'intero disco, rendendolo completamente irrecuperabile.

1.5.4 Supporti magneto-ottici

Figura 1.5: *Un Mini Disk a marchio Sony.*

Unico componente di questa famiglia è il MiniDisc di Sony, in tutte le sue varianti. La sua particolarità risiede nella tecnica di registrazione: da un lato un laser riscalda la superficie del disco rendendola suscettibile ai campi magnetici (a causa del raggiungimento del punto di Curie proprio del materiale che compone lo strato incidibile); nel frattempo, dal lato opposto del supporto, una testina magnetica incide il segnale digitale. In fase di lettura, invece, viene impiegato il solo laser, sfruttando l'effetto magneto-ottico di Kerr (MOKE): la luce polarizzata emessa (a potenza inferiore di quella utilizzata in scrittura) viene riflessa in maniera differente in base alla magnetizzazione del bit da leggere, potendo distinguerne il valore.

I supporti magneto-ottici e i dischi ottici condividono gli stessi problemi di deterioramento dei dischi ottici.

Formato	Substrato	Materiale	Tecnica di registrazione	Periodo
MD stampati	Policarbonato	Alluminio, vernice	Digitale	1992-2013
MD scrivibili	Policarbonato	Ferromagnetico (strato magneto-ottico), Alluminio (riflettente)	Digitale	1992-2013

Tabella 1.4: *Tipologie di supporti magneto-ottici digitali*

1.6 Copie conservative e copie d'accesso

Prodotto del processo descritto in 1.4, la copia conservativa viene definita come "l'artefatto preposto a essere conservato e mantenuto come *preservation master*, e pertanto da utilizzare solo in circostanze eccezionali" []. Dal lato pratico, inoltre, l'utilizzo di formati e codificatori ad alta qualità aumenta le dimensioni dei file risultato della digitalizzazione, rendendo la copia di difficile fruizione, sopratutto se richesta online. Per superare le limitazioni esposte è possibile quindi creare delle copie *di accesso*, cioè copie di qualità inferiore da affiancare all'originale, da mettere a disposizione dei fruitori per i loro scopi. Copie più snelle e pratiche agevolano così tutte le attività di ricerca sul documento, di cui è sempre possibile richiedere il master in massima qualità qualora servisse; le tecnologie di codifica audio e video odierne, però, consentono di ottenere un grado di dettaglio sufficiente a buona parte delle necessità dei fruitori, soprattutto se viene correttamente bilanciato il rapporto compressione/qualità.

Una copia conservativa digitale deve contenere, per essere completa, ogni dettaglio del documento fisico originale: oltre al segnale audio dovranno essere documentati tramite immagini e testi descrittivi gli aspetti visivi (come etichette e custodie, ma anche segni di degradazione del supporto) e olfattivi (odori di muffa o acidi, dovuti allo stato di decomposizione dei materiali), tutti codificati in un unico *documento unimediale* [], cioè la fusione di diversi *media* in unico flusso di bit. La struttura logica di tale documento è così formata:

- Un report dettagliato del contenuto della copia e del processo con cui è stata generata, nonché degli operatori coinvolti;
- Il segnale audio in uno o più file, in base alle caratteristiche e allo stato del documento;
- I metadati di primo livello, ovvero i *checksum* dei file audio appena citati;
- I metadati di secondo livello, cioè le specifiche tecniche dei formati di file utilizzati nella copia;
- La documentazione fotografica del supporto e di ciò che lo accompagna;
- Uno schema tecnico che illustri il processo di trasferimento in digitale.

Il formato audio utilizzato al CSC di Padova, come consigliato da [3], è il *Broadcast Wave Format* [1], campionato ad almeno 96kHz e 24bit di risoluzione. Il formato BWF, così come WAV da cui deriva, codifica il segnale in *Pulse Code Modulation (PCM)*, e non attua alcuna compressione dei dati; è quindi un formato *lossless* [2], che favorisce la qualità del segnale audio alle dimensioni del file su disco.

La documentazione video, invece, può essere codificata in un formato *lossy*, a patto che vengano scelte risoluzione e bitrate tali da permettere il riconoscimento di eventuali alterazioni del supporto durante la riproduzione. Un documentazione video adeguata dovrebbe poter fornire:

- Informazioni sullo stato fisico del supporto e su eventuali danni o deformazioni, così da poter distinguere le alterazioni intenzionali da quelle non intenzionali. Altre caratteristiche, come le giunture fra due nastri magnetici, devono essere ben riconoscibili.

- Informazioni sulle irregolarità della riproduzione, dovute ad assimetricità dei supporti o altre deformazioni: variazioni di velocità del supporto si riflettono nell'intonazione nel segnale audio analogico, con fenomeni di modulazione di frequenza definiti *wow* (se la modulazione è compresa tra 0.5 Hz e 6 Hz) e *flutter* (tra 6 Hz e 100 Hz). Alterazioni di questo tipo devono essere riconoscibili nella traccia video, per poterne determinare la loro origine e distinguerle da quelle già presenti nella registrazione originale.

- Istruzioni sulla riproduzione e sull'opera: principalmente presenti su nastri magnetici, sono delle annotazioni lasciate dall'artista sul supporto stesso per indicare alcuni punti di interesse. Ad esempio, è possibile trovare in alcune opere dei segnali per sincronizzare la riproduzione del nastro con altre entità, come un'orchestra.

Il report descrittivo deve raccogliere tutte le informazioni necessarie a comprendere il contenuto della copia conservativa, così come la sua composizione: saranno presenti, quindi, la lista dei file audio e video contenuti e la loro posizione, i relativi metadati e la descrizione tecnica del documento e del segnale originale (tipologia, formato di registrazione, ecc.).

I dati di verifica dell'integrità, rappresentati da una raccolta di checksum dei file contenuti nella copia, sono necessari per garantire la sua validità nel tempo. Anche i supporti digitali come hard disk e memorie flash sono soggetti a degradazione fisica, e questo potrebbe portare alla corruzione di parti o delle intere copie conservative in essi contenute; il confronto fra i checksum forniti e quelli calcolati in fase di verifica permette di identificare eventuali dati corrotti. Il metodo descritto in 1.4 prevede di allegare ai file il relativo *checksum* MD5, nonchè i *message digest* SHA1 e CRC32. Queste informazioni vengono inoltre conservate in una base di dati (insieme ai metadati) separata dall'archiviazione delle copie conservative, per maggiore sicurezza in caso di danneggiamento e perdita di uno dei due *storage*.

[1] Estensione del formato WAV, a cui aggiunge la possibilità di includere metadati nell'header del file.

[2] Un formato è detto *lossless* quando le informazioni codificate in questo formato sono recuperabili nella loro interezza dopo la decodifica. Al contrario, compressioni che eliminano parte del contenuto informativo sono definite *lossy*.

Capitolo 2

Stato dell'arte: PSKit

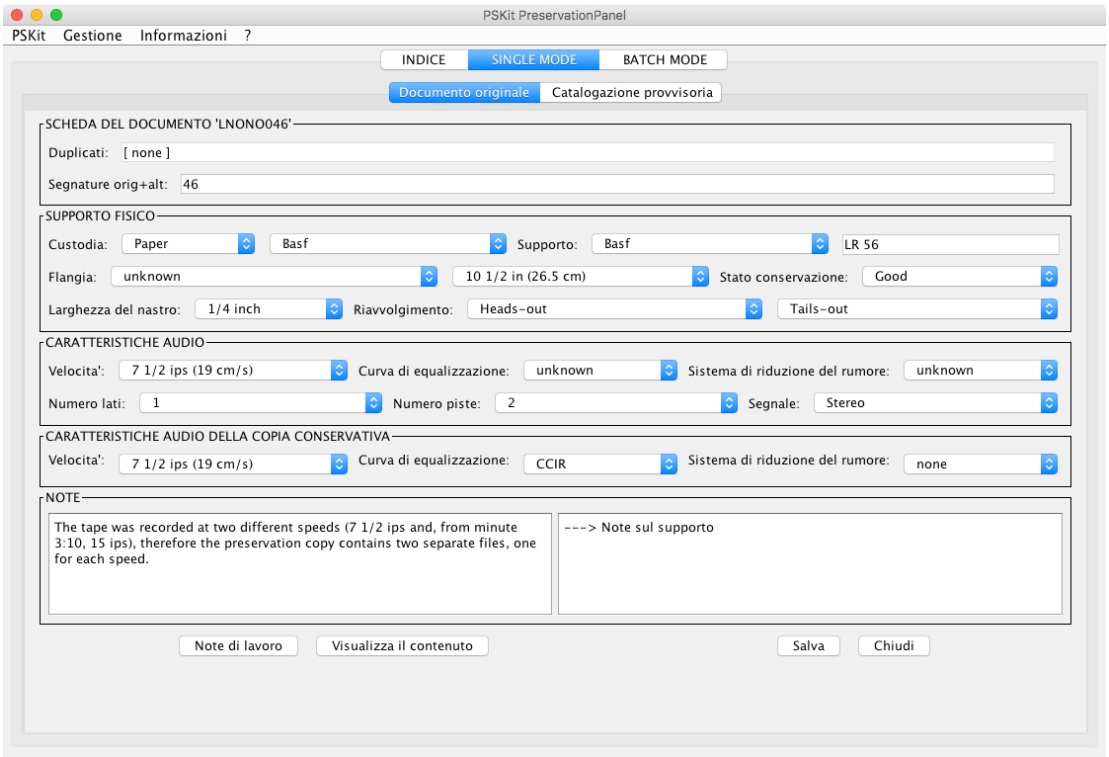


Figura 2.1: *Una schermata dell'applicazione.*

2.1 Introduzione

In 1.4 si è descritta la metodologia utilizzata presso il Centro di Sonologia Computazionale dell'Università di Padova, nata dall'attività di ricerca nell'ambito della conservazione di documenti sonori storici. Su questo metodo si basa *PSKit*, un pacchetto software atto alle operazioni di catalogazione e archiviazione. In questo capitolo se ne descriverà l'architettura e il funzionamento,

nonché si presenteranno le problematiche che hanno sollevato l'esigenza dello sviluppo di una nuova piattaforma alternativa.

2.2 Architettura

PSKit è un'applicazione desktop, scritta in Java SE, la cui UI è basata su Swing. L'elaborazione avviene in loco, all'interno dell'applicazione, mentre lo strato di persistenza, gestito da un DB MySQL, risiede su un server nella rete locale.

L'applicazione lavora esclusivamente nel file system locale, portando alla luce una prima problematica: i file generati dal processo di digitalizzazione, siano essi foto, audio o video, devono risiedere nella macchina in cui è in esecuzione PSKit. Questi sono solitamente file multimediali codificati in alta qualità, nonché acquisizioni di interi documenti sonori, che possono durare anche ben oltre i 60 minuti. L'insieme di questi due fattori comporta una dimensione occupata su disco nell'ordine delle decine di gigabyte, determinando un tempo molto lungo per trasferire l'intero pacchetto su tutti i PC su cui è in esecuzione il programma.

PSKit nasce come evoluzione dello strumento sviluppato nell'ambito del progetto *Gra.Fo.*, *Audiografo Preservation Panel*, per la gestione e creazione di copie conservative.

2.3 Il database

L'applicazione si appoggia al DBMS MySQL per gestire il tier di persistenza.

Il database è stato modellato sulle esigenze dell'applicazione ed espanso in tempi successi per l'aggiunta di nuove funzionalità.

Tutti gli attributi che prevedono un numero finito di valori sono implementati come *lookup-table*, cioè tabelle formate da un identificatore, un valore e, spesso, un riferimento al tipo di supporto a cui ogni riga è limitata. Questo permette una totale libertà di personalizzazione dei possibili valori, ma di contro incrementa il numero totale delle tabelle necessarie e dei vincoli per gestire le chiavi esterne.

In Figura 2.2 è riportato lo schema logico-relazione del DB, da cui sono state escluse tutte le *lookup-table*, per far risaltare le relazioni fra le principali entità.

Il database presenta alcuni difetti, sia concettuali che implementativi:

- Alcune tabelle mancano di una chiave primaria e del relativo indice, in quanto le righe vengono riferite tramite altri campi. Non utilizzando alcun indice, le prestazioni potrebbero risentirne, soprattuto con l'aumentare del numero totale di righe della tabella.
- Alcuni riferimenti a entità esterne mancano del vincolo di chiave esterna, rendendo possibile l'inserimento di identificatori non esistenti, o l'eliminazione dell'istanza riferita senza generare alcun controllo.
- I nomi scelti per le tabelle e i campi sono in alcuni casi non autoesplicativi. Per quanto non sia di per sé un problema in termini di funzionalità, complica leggermente le operazioni di mantenimento.

- Non è stato previsto un sistema di autenticazione e autorizzazione: tutti gli operatori registrati sono liberi da password e hanno accesso totale a tutte le funzionalità del sistema. Data la natura non distribuita del software, pensato per essere eseguito in una o poche macchine all'interno del laboratorio, questa scelta non porta a grossi rischi di sicurezza. Rimane comunque un problema di integrità dei dati: non sarebbe difficile per un operatore, infatti, modificare accidentalmente il lavoro svolto da terzi.

2.3.1 Struttura

Entità principale dello schema è *documorig*: contiene tutti gli attributi propri di un documento. Non viene fatta distinzione fra i diversi tipi di supporto, che confluiscono tutti in questa entità; questa informazione viene in ogni caso specificata in `tipo` (chiave esterna verso la *lookup-table* `tipo`).

Tutti gli attributi, quindi, sono in comune: dall'analisi della documentazione si evince che alcuni di questi possono assumere valori costanti, o derivabili da altri, dipendentemente dal tipo di supporto specificato. Il database non prevede alcuno di questi vincoli, delegando al software l'onere di generare gli eventuali valori corretti per ogni documento, senza richiederli all'utente. In Appendice 5 è riportata una tabella che espone nel dettaglio quanto implementato dal software.

Direttamente collegata a `documorig` è l'entità `cc`, che raccoglie le informazioni proprie di una copia conservativa, compresi i parametri di riproduzione utilizzati nel processo di digitalizzazione, espressi negli attributi: `eq` (equalizzazione), `nr` (riduzione del rumore) e `velocita` (velocità). I metadati dei file multimediali generati dal processo sono raccolti nelle tabelle `traccia`, `video` e `foto`, insieme al nome del file scelto.

Solo `cc` è legata a `persona` per specificare chi ha generato la copia conservativa: non è possibile tenere traccia di chi ha inserito le informazioni per un documento, nel caso non fosse la stessa persona.

Non è presente alcun sistema per registrare chi accede e modifica questo gruppo di entità in tempi successivi al primo inserimento: questo fatto, combinato alla mancanza di sistemi di autorizzazione e autenticazione, permette a chiunque, involontariamente o meno, di poter causare perdita di informazioni importanti relative a un documento (e relativa copia).

Un'istanza di `video`, inoltre, non è in alcun modo legata alla traccia a cui si riferisce: ogni riversamento del supporto deve essere infatti documentato con un apposita registrazione video, in modo tale da poterla sincronizzare con il relativo segnale audio. Spetta al software impostare il nome del file nelle due istanze allo stesso valore per poterli successivamente associare.

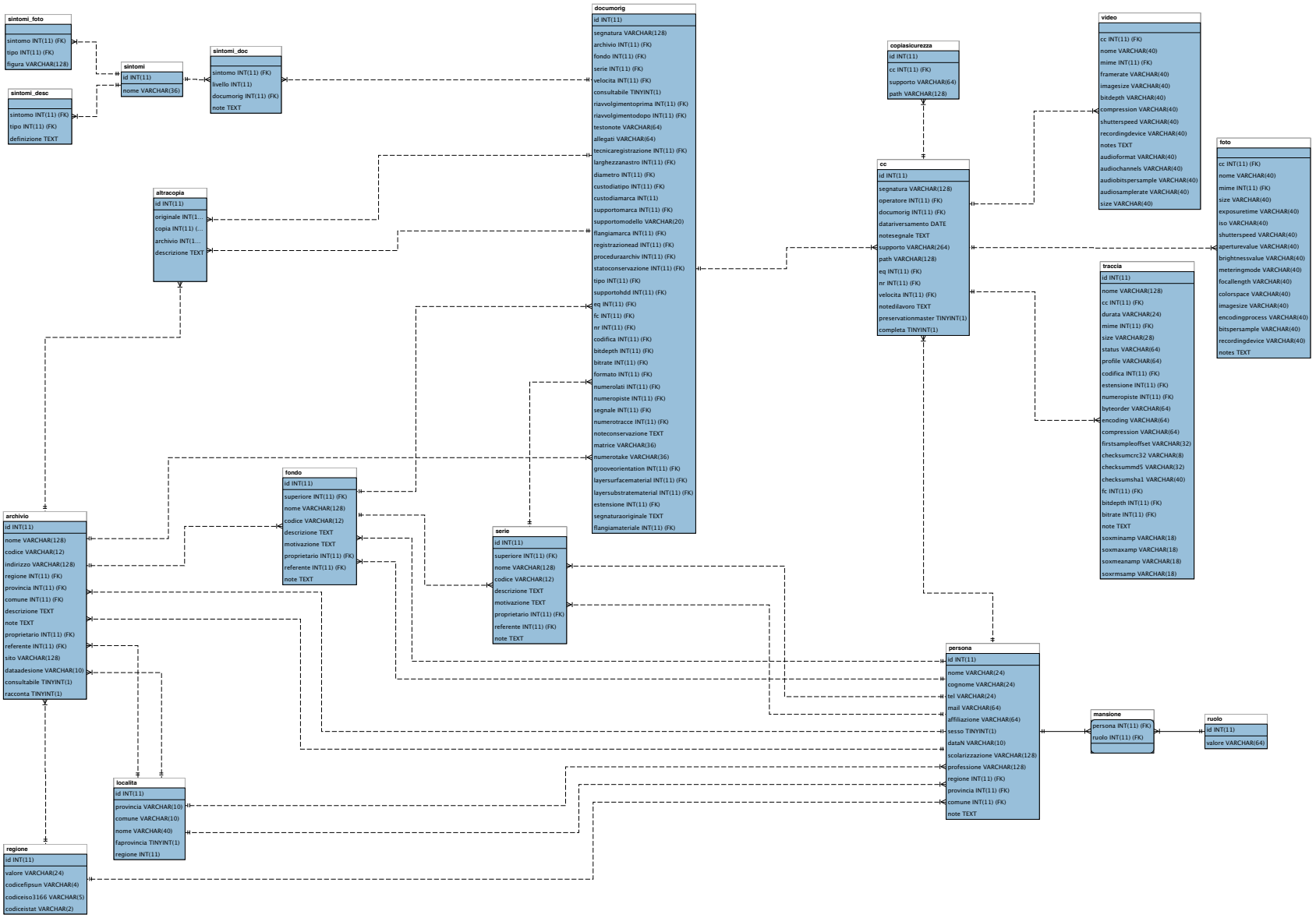


Figura 2.2: *Schema relazionale della base di dati senza tabelle di lookup.*

2.4 Il software

L'applicazione, come detto in 2.1, è scritta in Java, con l'ausilio della libreria grafica Swing per lo sviluppo dell'interfaccia utente. Non è stata impiegata nessun'altra libreria esterna, se non il driver JDBC per MySQL, che fornisce la connessione alla base di dati. Vengono però utilizzati alcuni software esterni, descritti in 2.5.

L'interfaccia è suddivisa in tre schede: *Indice*, *Single mode*, *Batch mode*.

Indice Fornisce una panoramica dei documenti creati, divisi per archivio, fondo e serie. L'elenco mostra solo i documenti creati dall'operatore attualmente utilizzato (e modificabile dal menu PSKit).

Single Mode Una volta selezionata una voce dall'indice si accede a questa scheda, che permette di compilare tutte le informazioni relative al documento e alla relativa copia. La scheda adatta il suo contenuto al tipo di supporto del documento selezionato, mostrando gli attributi opportuni. Oltre a questi, variano anche i campi relativi alla configurazione della macchina utilizzata per il riversamento (tipicamente equalizzazione, velocità, sistema di riduzione del rumore).

Batch Mode Scopo di questa scheda è raccogliere, verificare e generare tutti i file necessari per la creazione della copia conservativa, cioè quel pacchetto di documenti specificamente strutturato (vedi 2.4.1) per rappresentare l'intero contenuto informativo dell'opera in formato digitale. Tutte le operazioni vengono svolte per ogni copia in lavorazione (cioè quelle per cui esiste la relativa cartella nella macchina in uso), senza la possibilità di escluderne alcune o operare solo sul documento correntemente aperto. La scheda mette a disposizione i comandi per cercare in una specifica directory i file audio, video e foto da importare nella copia, estrarne e verificarne i metadati e generare checksum e report riassuntivo.

L'operazione di estrazione dei metadati dai file multimediali implementa un controllo per evitare di ripetere lavoro già svolto: se questi sono già presenti nel database, il file non verrà analizzato nuovamente. Se da un lato viene ridotto il tempo di esecuzione dell'operazione, dall'altro si limita la possibilità di sostituire uno di questi file in caso di errori o altre necessità: il controllo agisce infatti sul nome del file, non sul suo contenuto, rifiutando di eseguire l'analisi del nuovo file ononimo e sostituirla a quella precedente, non più valida.

Allo stesso modo anche la generazione del report PDF e dei file XML contenenti i metadati non viene ripetuta se già presenti, rendendo necessario eliminarli manualmente prima di poterli rigenerare.

2.4.1 Struttura delle copie conservative

A ogni copia conservativa viene riservata una cartella avente come nome la sua *signature*. All'interno di questa si trovano tre cartelle:

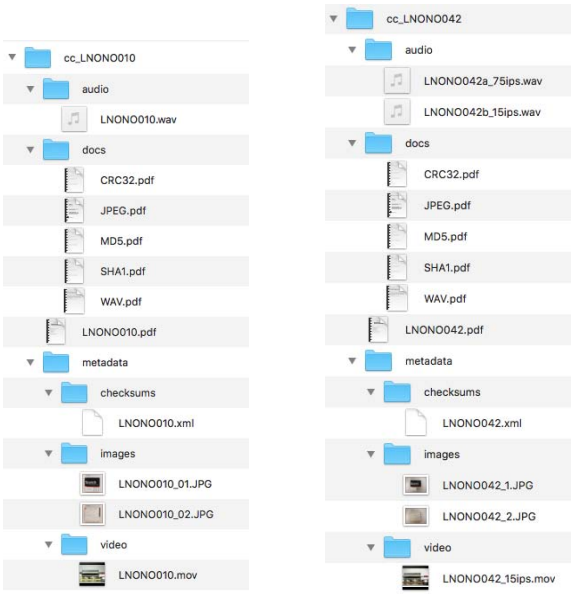


Figura 2.3: *Due esempi di copie conservative con i relativi file.*

***audio*:** contiene le tracce sonore, nominate con la *signature* del documento seguita da un eventuale lettera *a* o *b* per indicare il lato (solo per cassette e dischi fonografici) e una stringa di specifica introdotta da un *undescore*.

***docs*:** contiene i metadati di secondo livello, cioè dei file PDF riportanti le specifiche dei formati utilizzati.

***metadata*:** contiene i metadati di primo livello ed è a sua volta suddivisa in tre cartelle:

- ***images*:** contiene la documentazione fotografica del documento. Ogni immagine è nominata con la signature del documento e un numero progressivo, separata di un *underscore*.
- ***video*:** contiene le registrazioni video associate alle tracce sonore. I file, pertanto, sono nominati esattamente come le rispettive tracce in *audio*.
- ***checksums*:** contiene un file XML nominato con la *signature* del documento riportante i *checksums* di ogni traccia presente in *audio*.

Alcuni esempi di questa struttura di file sono visibili in Figura 2.3.

2.5 Tool esterni utilizzati

Il software usufruisce di strumenti opensource preesistenti per alcune operazioni di analisi ed estrazione dei metadati dai file multimediali che compongono la copia conservativa.

2.5.1 JHOVE

JSTOR/Harvard Object Validation Environment (JHOVE) è uno strumento di identificazione del formato, validazione e caratterizzazione di file multimediali. Rilasciato con licenza LGPL, è sviluppato in Java e può essere utilizzato *standalone* tramite GUI, da linea di comando o integrato come libreria in progetti terzi.

Nel caso di PSKit questo tool viene invocato tramite linea di comando per estrarre i metadati delle tracce audio delle copie conservative. I dati generati da JHOVE vengono salvati nella tabella `traccia` del database, insieme all'indicazione di validità del file.

Le principali informazioni estratte sono: formato del file, codificatore, *bit rate*, frequenza di campionamento, numero di canali e checksum CRC32, MD5 e SHA1

Un file viene considerato valido per PSKit quando risulta *well-formed*, cioè quando rispetta lo standard definito dal suo formato. Un esito negativo alla validazione potrebbe essere causato da errori nella generazione dello stesso (es: lunghezza del *payload* riportate nell'*header* errata, file troncati o corrotti), oppure da estensioni non standard del formato (es: metadati non previsti) applicate dal software utilizzato. Questo tipo di aggiunte non standard alle specifiche sono un rischio: queste, infatti, col tempo potrebbero essere superate o venire perdute (ad esempio per il cessato supporto della software house che le ha realizzate), rendendo i file incompatibili con i programmi di riproduzione e mettendo così a rischio la fruibilità della copia nel tempo.

Durante l'utilizzo dello strumento nell'ambito *PsKit* è emerso un bug che ne limita l'uso con le tracce più lunghe: i file WAV di dimensione superiore ai 2 GB vengono riconosciuti come invalidi, portando *JHOVE* a interrompere l'analisi, ritornando un errore. La specifica stessa di questo formato audio prevede ufficialmente una dimensione massima proprio di 2 GB, quando il campo del *header* che contiene la lunghezza del file potrebbe supportare fino al doppio (essendo un intero a 32bit *unsigned*). Nella *repository git* ufficiale del progetto[1] è presente un *pull request*[2] che risolve proprio questo problema, ma che gli sviluppatori non hanno ancora approvato.

2.5.2 exiftool

exiftool è uno strumento, scritto in Perl, che permette di estrarre e manipolare i metadati di diversi formati video e immagine. Prende il nome dalla specifica *Exchangeable image file format* (Exif) sviluppata dalla *Japan Electronic Industries Development Association* (JEIDA), che estende i formati JPEG e TIFF con l'aggiunta di tag per i metadati, utili per indicare diverse informazioni sullo scatto e sulla macchina che lo ha generato.

PsKit utilizza questo strumento per estrarre i metadati dalle foto scattate al supporto e dai video registrati durante la rimediazione. I dati generati vengono salvati nelle tabelle `foto` e `video` del database.

Le principali informazioni estratte dalle foto sono: modello della macchina fotografica, dimensioni in pixel, tipologia di compressione, apertura del diaframma e lunghezza focale dell'ottica, tempo di esposizione e ISO. Per i video, invece, vengono raccolti principalmente: dimensione

[1] `https://github.com/openpreserve/jhove`

[2] Una modifica del codice inviata da un utente senza terzo, che deve essere approvata dal proprietario della *repository* prima essere unita al resto del codice.

in pixel, codifica del flusso video, *framerate*, nome della videocamera, oltre a formato, bit depth e numero di canali del flusso audio.

Durante l'uso è emerso un problema di incompatibilità dello strumento con i file video catturati direttamente dalla telecamera tramite il software *Final Cut*[3] (come prevede la procedura impiegata dagli operatori): i metadati estratti sono errati, indicando 3 canali audio e una frequenza di campionamento pari a 1 Hz. Soluzioni possibili sono la ricodifica del file (spesso insostenibile in termini di tempo) o la sostituzione dello strumento con un'alternativa equivalente (con rispettiva riscrittura di una porzione di codice dedicata all'interpretazione dell'output).

2.6 Strumenti aggiuntivi: specifica dei sintomi

Oltre al software *standalone*, la suite PsKit fornisce anche una *web application* sviluppata *ad-hoc*, il cui scopo è la specifica dello stato di conservazione di un documento, la cui interfaccia è visibile in Figura 2.4. Tramite questa pagina web l'utente può visionare un report dettagliato contenente tutte le informazioni reperibili per ogni documento presente nella base di dati. Lo strumento, inoltre, permette di specificare per gli stessi documenti i sintomi da cui sono affetti, indicandone il livello di gravità da 1 a 5, oltre a eventuali note aggiuntive.

L'intera applicazione web è scritta in PHP, linguaggio di scripting eseguito lato server, che può generare pagine HTML standard dinamicamente, in base ai parametri della richiesta effettuata dal browser. Tutta l'elaborazione avviene quindi nel backend, sostenuto da un server Apache gestito dal Dipartimento di Ingegneria dell'Informazione (DEI). Gli script accedono allo stesso server MySQL dell'applicazione desktop, da cui attingono i dati generati proprio da quest'ultima.

L'accesso è regolato mediante un semplice meccanismo di autenticazione con un numero limitato di password *hard-coded* all'interno del sorgente: tutti gli utenti che dovranno accedere al servizio condivideranno la stesse chiavi. Come già analizzato in 2.3, infatti, l'architettura non prevede l'autenticazione degli utenti per l'accesso ai contenuti.

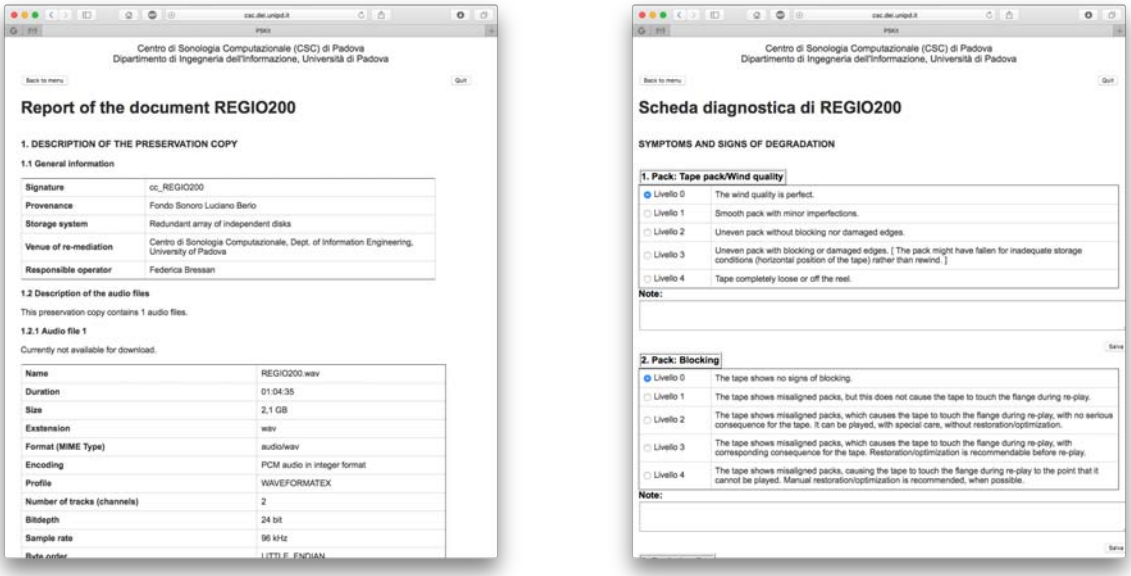
2.7 Interventi e analisi effettuate

Durante il lavoro di tesi, sono state apportate alcune modifiche agli strumenti sopra citati, al fine di risolvere alcuni bug e migliorare funzionalità e stabilità del progetto.

Buona parte degli interventi sono rivolti alla generazione del codice LaTeX per il report da allegare alle copie conservative: dalla correzione di alcuni errori di battitura, alla formattazione di immagini e tabelle, alla rimozione di informazioni ripetute e inserimento di altre, mancanti.

È stato scritto, inoltre, del codice aggiuntivo per estrarre dal database i metadati relativi alla macchina fotografica utilizzata per documentare i supporti, prima prefissati all'interno del codice (risultando potenzialmente invalidi in caso di cambio della macchina a disposizione del laboratorio).

[3] Software per la cattura e l'*editing* video sviluppato da Apple, compatibile esclusivamente con il sistema operativo *OS X* (ora *macOS*).



(a) **(b)**

Figura 2.4: *Interfaccia web per la specifica dei sintomi*

Più volte si è reso necessario intervenire manualmente nella base di dati per soddisfare le richieste degli operatori che ne facevano uso: sono state inserite manualmente diverse nuove voci fra le possibili scelte di marchi produttori, così come delle voci di velocità per alcuni tipi di supporto.

Altro problema simile è legato alle tracce audio e video durante la creazione della copia conservativa: una volta importati i file corretti, il software ne esegue l'analisi e ne inserisce il risultato nella base di dati; una volta inserita la riga, ogni successiva importazione di un file con lo stesso nome non verrà portata a termine, in quanto considerata già effettuata in precedenza. Non essendoci possibilità di eliminare quanto inserito, l'unica soluzione a una importazione errata è la cancellazione manuale della riga dalla relativa tabella nella base di dati. Il problema, abbastanza radicato nel codice, non è risolvibile senza richiedere una vasta riscrittura di parte dello stesso; pertanto si è consigliato l'uso di uno script di analisi standalone che utilizza gli stessi strumenti esterni utilizzati nel software. L'operatore, tramite questo strumento, può accertarsi della correttezza del file da importare; in questo modo verrà affidato al software solamente il file multimediale definitivo, senza la necessità di agire manualmente e ripetutamente nella base di dati.

Un bug presente nel software ha richiesto un ulteriore intervento di modifica del codice: nelle schermate dell'applicazione che richiedono l'inserimento di una data, il menù a tendina per la selezione nel mese mostrava solamente i primi sei valori (da Gennaio a Giugno) intervallati da voci nulle; il problema si è scoperto essere causato dalla procedura di estrazione delle voci dalla base di dati (più precisamente, dalla tabella *mese*): l'indice che indicava la loro posizione nella lista da mostrare nell'interfaccia era determinato dal loro *id*, ma questo, assumendo solamente

valori dispari (a causa dell'intervallarsi delle varianti in italiano e inglese), non permetteva di occupare tutte le 12 voci predisposte nell'interfaccia. Sarebbe stato possibile cambiare le chiavi primarie nella tabella ma, oltre a mettere a rischio l'integrità dei dati, avrebbe peggiorato il problema al cambio di lingua (se le voci italiane avessero avuto *id* da 1 a 12, quelle inglesi avrebbero assunto valori da 13 in poi, lasciando la lista nell'interfaccia completamente vuota); pertanto l'intervento ha coinvolto la *routine* di caricamento dei dati al fine di non utilizzare la chiave primaria come indice.

Questo intervento ha portato alla riscrittura di alcuni metodi di accesso alla base di dati, che effettuavano trasferimenti superflui tra strutture dati diverse, alcune delle quali implementate *ad-hoc*: si è preferito quindi usufruire di quanto offerto dalle librerie standard di Java, che, oltre a essere fortemente ottimizate, hanno permesso di ridurre il codice necessario, oltre che limitare le operazioni di travaso dei dati tra diversi contenitori.

2.8 Ristrutturazione della *suite*

Le problematiche esposte in questo capitolo hanno fatto emergere la necessità di ristrutturare l'intera *suite* di software in uso, a causa di alcune scelte progettuali e implementative che non permettono agli operatori di svolgere i loro compiti al meglio, qui di seguito riassunte:

- Impossibilità di eliminare archivi, fondi, serie, documenti, tracce audio e video
- Nomi dei documenti e delle copie conservative dettate dalla *signature*, limitata a 5 caratteri più 3 numeri.
- *signature* utilizzata come identificativo della copia (pur esistendo l'*id* numerico), rendendone impossibile la modifica. Se aggiunto all'impossibilità di eliminare gli elementi, si può notare come un nome errato generi una copia inutilizzabile, ma permanente.
- Impossibilità di elaborare una singola copia conservativa nella scheda *Batch Mode*
- Altri bug legati agli strumenti software utilizzati per l'estrazione dei metadati (e quindi non direttamente imputabili al PsKit)

In prima istanza si è pensato a una revisione di quanto attualmente esistente ma, nella prospettiva di coinvolgere e unire un numero di progetti paralleli operanti nell'ambito della conservazione di opere storiche musicali, si è propenso per una totale riscrittura della *suite*, a partire dalla base di dati. Nel prossimo capitolo saranno quindi esposte le informazioni raccolte e analizzate per definire i requisiti della nuova piattaforma.

Capitolo 3

Progettazione

3.1 Introduzione

3.1.1 Scopo

L'applicazione dovrà fornire un sistema di gestione, catalogazione e archiviazione di copie conservative di documenti sonori, provenienti da molteplici fonti. Diversi utenti, con ruoli diversi, avranno la possibilità di accedere contemporaneamente ai diversi progetti in corso, nonché lavorare su quelli a loro assegnati.

Ogni progetto potrà avere caratteristiche e requisiti diversi, come la formattazione del report, il nome da assegnare a ciascuna copia o l'aggiunta di dati solo a particolari documenti.

Infine, dovrà essere possibile esportare tutti i dati inseriti con i relativi file multimediali (foto, video, tracce audio) in una struttura di file ben definita e pronta per essere consegnata al committente.

3.1.2 Definizioni

Documento originale documento sonoro su supporto fisico, fonte della copia conservativa.

Copia conservativa Copia digitale del segnale audio di un documento, accompagnato da informazioni contestuali come foto e video, il cui scopo è conservare l'intero contenuto informativo dello stesso.

Archivio istituzione che fornisce il materiale e commissiona il lavoro di digitalizzazione e conseguente realizzazione di Copie Conservative.

Fondo collezione di Documenti con caratteristiche comuni (es: l'autore) su cui viene commissionato il lavoro di digitalizzazione e conservazione. L'organizzazione in Fondi è scelta dall'Archivio che possiede i Documenti.

Serie sottoinsieme di Documenti in un Fondo individuato da criteri interni (es: numero di scatola, lotto di numeri seriali).

Progetto contesto di lavoro associato a un Archivio o Fondo in cui alcuni utenti possono creare e gestire i Documenti e le Copie Conservative.

Utente persona registrata che, accedendo al sistema, può eseguire determinate operazioni in base ai permessi ottenuti.

3.2 Requisiti

Durante tutto il periodo di analisi e progettazione della nuova piattaforma si sono svolti degli incontri con i futuri utilizzatori, da cui sono emerse alcune richieste sulle caratteristiche e il funzionamento della stessa: dalla rielaborazione di quanto acquisito, unita allo studio della piattaforma già esistente (Capitolo 2), è stata stilata una lista di requisiti fondamentali, riportati nei prossimi paragrafi.

3.2.1 Requisiti funzionali

Il lavoro viene suddiviso tra i vari utenti sotto forma di Progetti. Ogni Progetto avrà degli utenti che ci lavorano, un referente e verrà associato a una o più Serie dello stesso Fondo, a cui apparterranno tutti i documenti trattati nel Progetto.

Le Serie e, di conseguenza, i Progetti, possono contenere Documenti tra loro disomogenei (tipicamente nel tipo di supporto, ma non solo).

Le opere fornite da un Archivio sono partizionate in più Fondi, che a loro volta sono partizionati in Serie. Deve esistere almeno una Serie per ogni Fondo, come un Fondo per ogni Archivio. Un Documento appartiene a un sola Serie. Ogni Documento potrà essere associato a una sola Serie e potrà essere contrassegnato come completato o meno, per poter conoscere immediatamente lo stato dei lavori.

Un Documento dovrà presentare degli attributi specifici per il suo tipo di supporto, oltre a quelli comuni a ogni caso. Questi attributi dovranno essere ben definiti a priori per ogni tipologia di documento e dovranno essere gestiti separatamente nella base di dati.

Ogni Documento dovrà poter contenere più File Set, ciascuno associato a un determinato insieme di parametri utilizzati per la riproduzione e il trasferimento in digitale. Ogni File Set potrà contenere più tracce sonore, cui può essere collegata una traccia video che mostri il supporto durante la riproduzione.

Ogni Utente registrato nel sistema può avere, per ogni Serie, un diverso livello di accesso ai Documenti (ogni livello estende i permessi di quello inferiore):

- Sola lettura dei documenti contenuti nella Serie
- Creazione e modifica dei File Set di un documento esistente
- Creazione e modifica di nuovi documenti all'interno della serie
- Definizione di nuovi Progetti

In parallelo, ogni Utente potrà avere un diverso permesso per accedere alle Analisi musicologiche dei Documenti contenuti in una Serie:

- Creazione, modifica ed eliminazione un'Analisi di cui è proprietario.
- Accesso in lettura a tutte le Analisi, qualunque sia il proprietario.

I progetti potranno essere creati e chiusi solamente dagli utenti che posseggono il relativo permesso.

Ogni Documento, indipendentemente dal suo Tipo di Supporto, potrà contenere delle informazioni aggiuntive che non rientrano negli attributi predefiniti.

Un Documento dovrà tenere traccia di quale Utente lo ha creato e quale ha effettuato l'ultima modifica. Dovrà essere possibile controllare l'accesso in concorrenza ai Documenti per evitare situazioni di perdita o inconsistenza dei dati.

3.2.2 Requisiti dell'interfaccia utente

L'interfaccia utente dovrà essere semplice e il più possibile immediata, visualizzando in primo piano tutti i progetti a cui l'Utente può accedere, seguiti dai quelli consultabili in sola lettura. In base al proprio ruolo, l'interfaccia dovrà visualizzare le azioni possibili in un determinato contesto. Ogni operazione potenzialmente distruttiva richiede la visualizzazione di un messaggio di conferma, avvertendo l'utente delle possibili conseguenze di tale operazione.

Insieme all'archiviazione, il progetto dovrà fornire la possibilità di accedere ai documenti per permettere all'utente di fruirne il contenuto nella sua interezza. Dovrà quindi essere possibile visualizzare le eventuali tracce video insieme al segnale sonoro, oltre a tutte le informazioni che le accompagnano, come foto e report.

Le componenti necessarie per permettere ai musicologi di effettuare delle analisi sono:

- Copie di accesso, per agevolare la fruizione a chi accede da remoto
- Possibilità di salvare i dati generati dai musicologi in maniera strutturata
- Un'interfaccia che visualizzi l'intero contenuto della copia, insieme a strumenti atti a facilitare le operazioni di analisi.

Quanto richiesto al primo punto può essere gestito dal software su comando dell'operatore: ogni qualvolta costui contrassegni un opera come completata, il server potrà eseguire le operazioni di transcodifica dei file forniti (o pianificarne l'esecuzione in un momento successivo, di minore carico). Alternativamente potrà essere l'operatore stesso a eseguire queste operazioni e trasferire le copie d'accesso realizzate alla posizione corretta.

Per soddisfare il secondo punto, i dati saranno conservati nella base di dati insieme a un puntatore ai documenti cui si riferiscono. Ogni analisi, quindi, sarà associata a un specifico File Set e dovrà essere composta da un'insieme di Unità, ciascuna descritta da:

- Un insieme di coppie tipo-valore, chiamate *attributi*, in cui il tipo è limitato a un numero finito di possibilità. I tipi accettabili per un attributo potranno variare in base all'appartenenza del relativo documento.
- Una lista di *eventi*, cioè descrizioni di quanto avviene in un intervallo di tempo all'interno della traccia in analisi.

L'interfaccia, che dovrà essere accessibile pubblicamente, dovrà permettere all'utente di generare *analisi*, *unità*, *eventi* e *attributi*. Tra gli strumenti aggiuntivi, potrà esserci la possibilità di mostrare la forma d'onda del segnale audio in riproduzione, sincronizzata con l'eventuale registrazione video.

In aggiunta a questi strumenti dovrà essere possibile inserire nel sistema, per ogni traccia audio e video, dei *punti di interesse* che evidenzino un determinato avvenimento a un preciso istante temporale: l'inizio dell'opera nella traccia, una giunta del nastro in una bobina, un rumore o una distorsione dovute al degrado fisico o chimico del supporto, ecc. Ogni musicologo dovrà, inoltre, poter segnalare la validità o meno di un punto di interesse incontrato durante l'ascolto.

L'interfaccia per l'analisi dovrà implementare i seguenti criteri di controllo sui dati:

Inizio e fine evento: vengono considerati validi se:

- non sono vuoti
- sono in formato di tempo valido, come `hh:mm:ss.sss` (con `sss` parte decimale dei secondi, per avere una risoluzione pari a 1ms)
- non superano la lunghezza della traccia
- *Inizio evento* è antecedente a *Fine evento*

Ogni cambiamento nei valori di inizio e fine evento dovrà innescare la validazione di entrambi i campi. La relativa componente *backend*, invece, dovrà soddisfare i seguenti requisiti:

Tipo attributo: deve essere definito, altrimenti l'attributo dovrà essere ignorato.

Attributi: possono coesistere attributi dello stesso tipo per la stessa *unità*, purché il loro *valore* sia diverso.

3.3 Ruoli

Amministratore: Crea nuovi Utenti, gestisce l'infrastruttura e ha accesso in scrittura a tutte le risorse del sistema. Può assegnare permessi ad altri Utenti.

Operatore: può creare nuovi Documenti. Può accedere in scrittura a tutti i Documenti delle Serie per cui ha il permesso, anche se creati da un altro Operatore. Può segnare un Documento come completato, e quindi pronto per l'esportazione.

Operatore di digitalizzazione: Utente che si occupa del riversamento dei Documenti. Può creare e popolare i File Set di un Documento già esistente.

Utente esterno: Persona non facente parte del laboratorio che ha accesso in lettura ai Documenti completati per poterli analizzare tramite l'interfaccia di accesso.

3.4 Progettazione della base di dati

3.4.1 Prima versione

Data l'elevata dinamicità richiesta dai requisiti, è stata progettata una prima versione della base di dati che permettesse una totale variabilità delle informazioni inserite, senza vincolare i dati a schemi prestabiliti, offrendo così la possibilità di effettuare modifiche anche sostanziali alla struttura dei documenti senza dover modificare la struttura della base di dati stessa.

Per permettere ciò, nello schema sono presenti due entità: *Documento* e *Proprietà*. Ogni *Proprietà* rappresenta un attributo che un *Documento* può assumere ed è definita da un nome e un tipo, sia esso *numerico*, *stringa* o *enumeratore*. In quest'ultimo caso, i possibili valori che *Proprietà* può assumere vengono espressi con una relazione uno-a-molti verso, appunto, *Possibile Valore*.

Ogni istanza di *Documento* può essere legata a una lista di *Proprietà* e, per ognuna, specificare il valore che questa assume. Il concetto viene implementato con una relazione molti-a-molti che implementa l'attributo *Valore*.

Allo stesso modo sono stati implementati i *Riversamenti* e le relative *Configurazioni*, cioè quegli attributi che descrivono le impostazioni utilizzate per la riproduzione: tra le due entità sussiste una relazione molti-a-molti che espone l'attributo *valore*. In questo caso, però, *Configurazione* può essere solamente del tipo *enumeratore*, per cui è legato con una relazione obbligatoria uno-a-molti a *Possibile Valore*.

Alcuni di questi possono essere considerati accettabili solo se a una determinata *Proprietà* è stato assegnato un preciso valore. Questo legame di dipendenza è rappresentato da un'associazione circolare facoltativa in *Possibile Valore*. Questi vincoli non coinvolgono le *Proprietà* cui i valori sono legati: non è necessario esplicitare la loro appartenenza nella relazione di dipendenza, in quanto ogni *Possibile Valore* può essere associato a una e una sola di esse.

Ogni *Riversamento* è associato a diverse tracce audio, espresse con l'entità *Audio*, cui, a sua volta, è legato un *Video*, se presente. La documentazione fotografica è espressa dall'entità *Foto* ed è associata a *Documento*, in quanto non dipende dalla configurazione utilizzata per riprodurre il supporto.

La gerarchia di appartenenza dei *Documenti* è descritta dalle entità *Archivio*, *Fondo* e *Serie*, legate tra di loro da associazioni uno-a-uno per soddisfare il requisito di esistenza delle stesse. Ogni istanza di *Documento* è associata a una singola *Serie*, che a sua volta sarà assegnata a un singolo *Progetto*. Ognuno di essi può quindi raccogliere più *Serie*, con il vincolo esterno di appartenenza allo stesso fondo, coprendo così tutti (o parte) dei documenti forniti da un archivio, a prescindere dalla loro organizzazione interna.

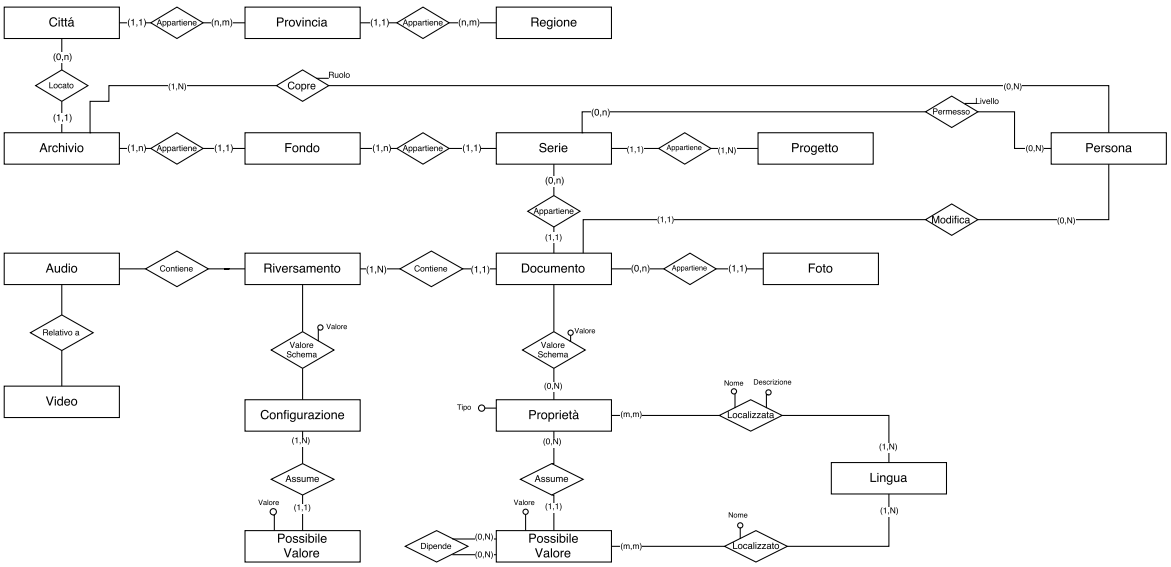


Figura 3.1: *Versione preliminare della schema ER per la prima proposta. Vengono mostrati solo alcuni attributi significativi al concetto di proprietà dinamiche.*

Il controllo di conformità a uno schema predefinito di proprietà per ogni documento viene lasciato al software che si appoggia a questa base di dati: è così possibile, qualora necessario, aggiornare i valori possibili per un enumeratore, definire nuove tipologie di supporto o modificare quelle già esistenti senza invalidare i documenti inseriti precedentemente.

In Figura 3.1 è rappresentata l'ultima bozza dello schema ER relativo a questa prima proposta. Si è reso necessario infatti ristrutturare e rivedere l'intera struttura dello schema per soddisfare una richiesta specifica di completa indipendenza tra la componente di persistenza e il software che l'avrebbe utilizzata. Caratteristica fondamentale di questa proposta è, infatti, la profonda dipendemza con il software, che avrebbe dovuto farsi carico di un buon numero di vincoli per permettere una completa dinamicità delle informazioni senza perdita di consistenza nei dati inseriti. Data la natura variabile del team di sviluppo del progetto (in quanto composto principalmente da tesisti presso il CSC, la cui permanenza media è nell'ordine di qualche mese), si è preferito fissare il maggior numero di vincoli a livello di base di dati, in modo tale da salvaguardarne il contenuto da malfunzionamenti o caratteristiche non ancora complete o consolidate del software in sviluppo.

3.4.2 Seconda versione

Lo schema ER è stato rimodellato per permettere una maggiore indipendenza dagli strati software superiori, affidando la gestione e il mantenimento della consistenza e dell'integrità dei dati completamente al database. Si è deciso così di abbandonare l'approccio dinamico per gli attributi di un documento, creando entità separate per ogni tipologia di supporto.

In Figura 3.2 è riportato lo schema finale, in cui non vengono mostrati tutti gli attributi di ciascuna entità per ragioni grafiche.

La gestione di archivi, fondi e serie rimane invariata, mantenendo la relazione di appartenenza fra le entità *Document* e *Series*; permane il vincolo per cui, qualora *Archive* non sia suddiviso in *Fonds* e/o *Series*, si crei un'istanza unica di questi che copra tutto il contenuto del livello superiore. Allo stesso modo anche *Project* non cambia, rimanendo legato a più *Series*, purché appartenenti allo stesso ramo della gerarchia.

La differenza sostanziale risiede nella gestione dei documenti: rimane l'entità *Document*, cui sono assegnati gli attributi comuni, che viene ora specializzata in diverse entità figlie, una per tipo di supporto, ognuna con i relativi attributi specifici. Per ogni riversamento del supporto effettuato è presente l'entità *File Set*, che funge da contenitore per le tracce audio e video e in cui sono raccolti gli attributi che descrivono i parametri utilizzati per riprodurre il documento. I tipi di supporto esplicitati nello schema sono, come già analizzati in 1.5:

- Audiobobina di nastro magnetico, analogica;
- Audiocassetta di nastro magnetico, analogica;
- Disco fonografico, analogico;
- DAT (*Digital Audio Tape*), nastro magnetico con codifica digitale;
- CDA (*Compact Disk - Audio*), disco ottico con codifica digitale;
- Mini Disk, tecnologia magneto-ottica con codifica digitale;
- Documento Digitale, un generico contenitore di dati digitali, come hard disk o chiavette.

L'entità *Document* è legata con un'associazione uno-a-molti a *Photo* e *Attachment*, che rappresentano rispettivamente la documentazione fotografica e gli allegati che accompagnano il supporto, solitamente documenti cartacei che vengono digitalizzati tramite l'uso di uno *scanner* in quanto non adatti a essere fotografati (per via, ad esempio, di scritte di piccole dimensioni o sbiadite dal tempo).

Ogni tipologia di supporto analogico prevede dei parametri di riproduzione diversi, in alcuni casi vincolati tra di loro. Dopo un'accurata analisi dei diversi casi, è stata studiata una soluzione che permettesse di unificarli nella stessa entità. Gli attributi coinvolti sono *velocità* (sia lineare che angolare), *equalizzazione* e *noise reduction*: mentre quest'ultimo è indipendente, i rimanenti sono legati da un vincolo dettato dagli standard utilizzati nel corso degli anni. Si è scelto pertanto di implementare un'ulteriore entità, chiamata *Preset*, che rappresenti ogni combinazione valida di velocità di avanzamento ed equalizzazione, legandolo alla relativa tipologia di supporto. In Tabella 3.1 sono riportati i vincoli definiti: si può notare come bobine e cassette siano limitate nella scelta della coppia velocità-equalizzazione, mentre tutti i supporti digitali non prevedano alcuno di questi valori (in quanto propri dei segnali analogici); un caso particolare è quello dei dischi fonografici: durante la fase di ricerca non è emersa una lista ben definita di vincoli, pertanto si è deciso di separare la velocità di rotazione, specificandola in *File Set*, per porre in *Preset* tutte le possibili equalizzazioni per questo tipo di supporto, impostando a un valore nullo la relativa velocità lineare.

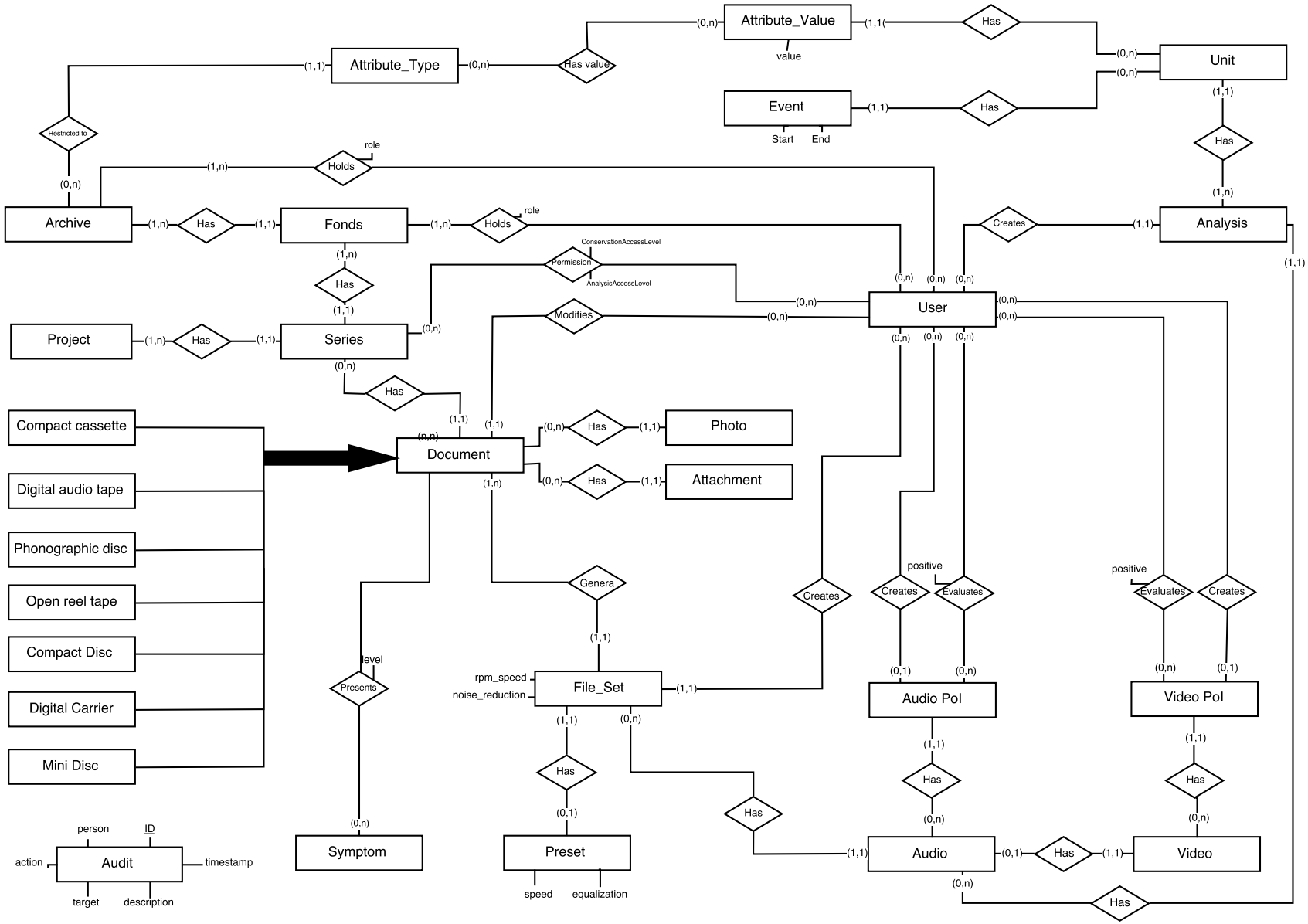


Figura 3.2: *Schema ER della base di dati, seconda versione.*

	Preset		Velocità (RPM)	Noise reduction
	Velocità lineare	**Equalizzazione**		
Audiobobina	Y		N	libera
Audiocassetta	Y		N	libera
Disco Fonografico	null	Y	Y	libera
Supporti Digitali	N		N	N

Tabella 3.1: *Configurazione degli attributi in FileSet per i diversi tipi di supporto. Notare come per i dischi fonografici i valori di Preset presentino una velocità lineare nulla, dovendo specificare separatamente quella di rotazione.*

In quest'ultima entità sono definite delle informazioni aggiuntive che descrivono e caratterizzano i diversi preset di equalizzazione. Possono essere specificate in tre forme, tra di loro equivalenti:

Costanti di tempo: espresse in secondi, caratterizzano l'equalizzazione RIAA (da *Recording Industry Association of America*) utilizzata nei dischi fonografici in vinile.

***Bass Turnover + Treble Turnover*:** frequenze in Hz che indicano il punto di transizione fra, relativamente, bassi/medi e medi/alti.

***Bass Turnover + Rolloff*:** il *rolloff*, espresso in dB, esprime l'attenuazione da applicare a 10kHz (punto di taglio delle alte frequenze)

Oltre a queste è descritta la presenza di un filtro di *Low Frequency Shelving* tramite un apposito *flag*. Tutte le rappresentazioni sono presenti all'interno dell'entità come attributi, raggruppando i due *Bass Turnover* in un unico elemento comune. Si potrà quindi implementare la possibilità di specificare solo una delle possibilità e lasciare al software, o alla base di dati, la conversione e popolamento le rimanenti grazie alle seguenti formule:

$$f_{turnover} = \frac{10^6}{2 * \pi * t} \qquad f_{treble} = \sqrt{\frac{10000^2}{10^{-\frac{rolloff}{10}} - 1}}$$

Nello schema sono presenti, inoltre, tutte le componenti che definiscono le analisi musicologiche, così come descritto nei requisiti: entità centrale è *Analysis*, legata a un'istanza di *Audio*, a cui appartengono più istanze di *Unit*. Queste, a loro volta, sono legata uno-a-molti con *Event*, che descrive un intervallo temporale all'interno della traccia. Ogni *Attribute Type*, invece, è legato a un singolo *Archive*, per indicarne l'appartenenza e limitarne quindi l'uso. *Attribute*, infine, lega un tipo all'unità, specificandone il valore assunto: questa entità non utilizza come chiave primaria la combinazione di chiavi esterne verso *Attribute Type* e *Unit* poiché è permesso avere due attributi dello steso tipo all'interno della stessa unità. Per rappresentare i punti di interesse (*PoI, Point of Interest*) sono presenti le entità *Audio PoI* e *Video PoI*, associate ciascuna a un'istanza delle rispettive tracce multimediali. Per ogni punto di interesse è possibile specificare una valutazione da parte di un'istanza di *User* grazie all'associazione tra queste entità, che espone un apposito attributo booleano, *positive*.

L'entità *User* rappresenta un qualsiasi agente operante nel sistema, sia esso umano o software, cui possono essere assegnati diversi permessi di accesso in lettura e scrittura ai documenti, ai loro riversamenti e alle relative analisi musicologiche, così come definito in 3.2.1. Questi permessi sono specificati nell'associazione con *Series*, che presenta due attributi: il livello di accesso alla sezione di conservazione (documenti e fileset) e a quella di analisi (punti di interesse, analisi musicologica); in *User*, inoltre, è presente un *flag* per specificare il ruolo di *amministratore*, che permette di ottenere qualunque permesso nell'intero sistema.

Infine, è presente un'entità isolata, chiamata *Audit*, il cui scopo è raccogliere tutti gli eventi di modifica, cancellazione e creazione che coinvolgono il resto dello schema. Ogni istanza descrive quindi: l'avvenimento, chi (*person*) ha eseguito un'azione (*action*, può assumere i valori *create*, *update*, *delete*) e su quale entità (*target*, stringa rappresentante il nome dell'entità conivolta). La base di dati, non gestendo le procedure di autenticazione e autorizzazione, non può conoscere l'utente che ha generato un evento di modifica, pertanto la gestione di questa entità viene affidata completamente al software.

Capitolo 4

Implementazione

4.1 Tecnologie utilizzate

4.1.1 JSON

JSON (*JavaScript Object Notation*) è un formato adatto all'interscambio di informazioni fra applicazioni di rete. Come specifica l'acronimo, si tratta di un *subset* della sintassi JavaScript, in particolare quella relativa alla definizione di array e vocabolari (o array associativi).

Questo standard permette di codificare in un testo leggibile da umani e macchine delle informazioni complesse, anche organizzate in gerarchie. È possibile specificare i seguenti tipi di dato:

- `null`, un elemento nullo;
- booleani, tramite i valori `true` e `false`;
- numeri interi e decimali;
- stringhe di testo, racchiuse da doppi apici (contrariamente a JS che supporta anche gli apici singoli);
- array, una lista di dati separati da virgole e racchiusi in parentesi quadre;
- array associativi, una lista di coppie `chiave: valore` separate da virgola e racchiuse in parentesi graffe, dove la chiave è una stringa e il valore è un qualsiasi dato, compresi gli array associativi.

Il server, che si occupa di gestire tutte le operazioni, comunica via HTTP, ma non fornisce direttamente le pagine HTML da visualizzare: a ogni richiesta risponde, infatti, con delle informazioni nel formato JSON. In questo modo si provvede a separare il layer di presentazione da quello applicativo, rendendoli completamente indipendenti nella loro implementazione.

4.1.2 REST

REST (*REpresentational State Transfer*) è una tipologia di architettura software per sistemi client-server, il cui principio fondamentale è la separazione degli compiti fra le componenti distribuite del sistema, in modo da semplificarne l'implementazione eliminando ogni interdipendenza tra le stesse.

L'approccio proposto prevede, quindi, che il sistema venga progettato seguendo i seguenti principi:

- I dati e le funzionalità dell'applicazione devono essere rappresentate come risorse;
- Ogni risorsa deve essere univocamente identificabile e indirizzabile, solitamente tramite l'uso di URL;
- Le risorse devono essere accessibili attraverso un'interfaccia comune tra le varie componenti distribuite, che permetta di definire delle operazioni eseguibili su ciascuna.

Solitamente REST viene associato a HTTP, attraverso il quale vengono trasferite le rappresentazioni delle risorse. Questo protocollo risulta ottimale per l'implementazione di un sistema REST in quanto, oltre a essere estremamente diffuso nel *world wide web* e utilizzato per trasferire "documenti" locati a determinati indirizzi URL, ogni sua richiesta può essere completamente indipendente dalle altre e può specificare una determinata operazione (o metodo) da eseguire.

I metodi definiti dallo standard HTTP sono spesso utilizzati, quindi, per indicare al server come manipolare la *risorsa*:

GET: richiede una risorsa o una funzionalità

POST: invia al server una nuova risorsa, solitamente per aggiungerla a quelle esistenti e assegnarle un nuovo identificatore

PUT e PATCH: specificano al server le modifiche da effettuare su una risorsa esistente.

DELETE: richiede la cancellazione di una risorsa.

Seppur spesso associato a HTTP e XML o JSON, l'approccio REST è puramente concettuale e implementabile con qualsiasi tecnologia e protocollo ne permetta di soddisfare i requisiti.

Un sistema che segue i principi dettati da REST viene comunemente definito *RESTful*.

4.1.3 HATEOAS

HATEOAS (*Hypermedia As The Engine Of Application State*) è un vincolo del paradigma REST che si basa sull'esprimere le relazioni fra risorse attraverso link ipermediali (URI di risorsa). È possibile così interagire con il server solamente con gli ipermedia forniti, limitando così la necessità di conoscere priori il servizio offerto.

In questo modo il client non sarà costretto ad avere a priori tutti gli URL delle risorse esposte dal server, ma partendo dal cosiddetto *entrypoint* e conoscendo le relazioni che le legano, potrà

accedere a tutte le informazioni di cui necessita. Se implementato correttamente nel client, questo approccio può essere visto come un livello di astrazione posto al di sopra della comunicazione HTTP con il server: l'accesso alla risorsa è completamente trasparente al protocollo di trasporto e al formato dei dati scambiati, che possono essere modificati o espansi lato server senza perdita di funzionalità lato client. Questo permette una maggiore mantenibilità del progetto nel lungo periodo [1].

4.1.4 JSON HAL

HAL (*Hypertext Application Language*), nella sua variante JSON, è una specifica che descrive come una risorsa (o documento) codificata in JSON debba presentare le relazioni verso altre risorse. Se associato a un giusto design delle risorse esposte, può essere considerato come un'implementazione del vincolo *HATEOAS* nell'ambito di un API *REST*: la specifica infatti è più generica e descrive solo la sintassi con cui esprimere una risorsa e le sue relazioni, o *link*. In particolare, *HAL* prevede che una risorsa sia composta da:

- *Link* ad altre risorse,
- Eventuali risorse incorporate (*embedded*), cioè un insieme di sotto-risorse contenute al suo interno,
- Uno stato, rappresentato dai dati JSON che la descrivono

Un *link*, invece, è definito da:

- Un URI di destinazione che identifichi la risorsa a cui punta,
- Un nome, definito *relation* o, abbreviato, *rel*
- Alcuni attributi che lo descrivano, ad esempio un flag per indicare se l'URI è un template, o se la risorsa puntata è dichiarata come deprecata.

Come descritto in 4.1.11, il software sviluppato utilizzerà *HAL* per descrivere le relazioni fra entità e rendere le API offerte esplorabili.

4.1.5 Postgres SQL

Sviluppato da *PostgreSQL Global Development Group* e rilasciato su licenza libera BSD, *PostresSQL* è un RDBMS (*Relational DataBase Management System*) che implementa molte funzionalità aggiuntive rispetto ai concorrenti, tra cui *MySQL*: fra queste risaltano la possibilità di dividere una base di dati in *schemi* (concetto simile ai *namespace* in Java), il supporto ai tipi di dato enum e array e l'estensibilità tramite diversi linguaggi di programmazione.

[1] http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven

4.1.6 Spring

Spring è una collezione di progetti alla cui base si presenta un framework per lo sviluppo rapido di servizi Java. I vari progetti figli espandono le funzionalità del framework base, semplificando diversi aspetti dello sviluppo.

Caratteristica importante del framework è l'implementazione del pattern IoC (*Inversion of Control*) tramite *Dependency Injection*, che solleva il programmatore dal compito di creare e gestire i riferimenti alle istanze dei diversi servizi implementati attraverso l'uso di un apposito componente denominato *injector*. Con questa tecnica sarà il framework stesso ad inizializzare i vari oggetti, fornendo loro le dipendenze richieste. Scompare così dal codice applicativo ogni logica di inizializzazione. Tramite la configurazione (tramite file XML o specifiche classi Java) è possibile specificare le caratteristiche di ogni singolo componente (chiamato *bean*).

Unendo questo approccio con l'utilizzo di interfacce comuni, è possibile configurare facilmente un applicazione senza dover modificare l'implementazione della sua logica. Ad esempio l'accesso ad una base di dati è astratto da un'interfaccia specifica, che può essere utilizzata similarmente qualunque sia la sorgente (*MySQL*, *H2*, *PostgreSQL*, o tutto ciò che JDBC supporta); quale implementazione usare per questa interfaccia viene definita in un'apposita classe di configurazione, separatamente dal resto.

Spring Framework inoltre raccoglie in un ambiente comune ed astrae diverse funzionalità, dall'accesso diretto a JDBC, allo scheduling di attività, alla gestione asincrona di queste, al logging e molto altro. È quindi possibile adattare l'implementazione di ognuna di queste in base alle necessità dell'applicazione, scegliendo di usare delle implementazione più semplici (sfruttando gli strumenti presenti nelle API Java), o altre, più potenti, ma dipendenti da librerie sviluppate da terzi.

La documentazione fornita è estensiva e molto chiara, e grazie alla vasta diffusione del progetto non è difficile reperire materiale a riguardo, grazie alla *community* e agli stessi sviluppatori attivi in prima persona.

4.1.7 Spring Boot

Un sotto progetto molto interessante delle famiglia *Spring* è *Spring Boot*, che, implementando un sistema di *auto-discovery* dei *bean* presenti nel codice, permette allo sviluppatore di ridurre ai minimi termini la configurazione necessaria (cioè quel codice atto a specificare quali specifiche implementazioni fornire ai servizi che richiedono l'uso delle relative interfacce), pur rimanendo estremamente flessibile e personalizzabile. In questo modo è possibile concentrarsi sull'implementazione della *business logic*.

Altra caratteristica importante è la presenza di un server *Tomcat* integrato direttamente nel *package* compilato, che viene configurato appositamente per l'applicazione. In questo modo è possibile generare un server standalone che può essere eseguito con un normale *JAR* (*Java ARchive*, pacchetto eseguibile di codice bytecode Java, unito ad eventuali librerie e risorse utilizzate) senza richiedere dipendenze esterne, rendendo il ciclo di sviluppo estremamente agile.

4.1.8 Spring MVC

Il core di Spring, pur essendo pensato per il web, implementa solo uno strato di utilità comuni. Per poter gestire delle richieste provenienti dai client tramite protocollo HTTP si può utilizzare il progetto *Spring MVC*.

Il pattern MVC, acronimo di *Model, View, Controller*, è un *design pattern* in grado di separare la logica di separazione da quella di *business*, definendo tre tipologie di componenti:

Model: fornisce i metodi per accedere e manipolare i dati

Controller: riceve ed esegue i comandi impartiti dall'utente, ponendosi di fatto fra i modelli dei dati e le viste.

View: visualizza i dati ricevuti dal controller e permette l'iterazione con l'utente.

A discapito del nome, è possibile implementare anche solo alcune parti dei questo pattern: ad esempio, un server REST non avrà bisogno di gestire le *View* (intese come pagine HTML generate da template), in quanto ritornerà una rappresentazione delle risorse in puro testo; *Spring MVC*, pur implementandoli, non obbliga l'uso di *template engine* o simili, lasciando al programmatore la scelta di cosa restituire alla richiesta di un client.

Allo stesso modo la gestione dei modelli viene delegata all'implementazione del programmatore, o ad altri progetti (come quelli della famiglia *Spring Data*).

Anche per la gestione dei *Controller* e, di conseguenza degli URL a cui il server risponderà, viene lasciata totale libertà al programmatore, fornendo come sempre gli strumenti necessari per una facile implementazione.

Il vantaggio di questo modulo è la completa astrazione di tutto il protocollo HTTP utilizzato: non sarà necessario gestire le richieste provenienti da un *socket TCP*, né dover interpretare gli *header* della richiesta inviata o generare quelli della relativa risposta. Qualora ce ne fosse bisogno, però, è possibile manipolare queste informazioni per implementare funzionalità specifiche. Altro vantaggio è la gestione *multi-threaded* delle richieste: durante l'elaborazione di una richiesta, il software potrà accettare nuove connessioni da altri client, senza doverli lasciare in attesa.

4.1.9 Spring Security

Per poter gestire in maniera ottimale ed altamente flessibile il controllo di accesso nella propria *business logic* è possibile affidarsi a *Spring Security.*

Questo progetto si prende carico di gestire tutte le caratteristiche necessarie per la gestione dell'accesso tramite autenticazione e autorizzazione, lasciando totale libertà di personalizzazione per venire incontro alle necessità dell'applicazione.

Integrandosi con altri progetti della famiglia, *Security* ne espande le funzionalità con *feature* relative alla sicurezza e alla gestione dell'autenticazione. Ad esempio, se è presente *String*

MVC, *Security* aggiungerà la possibilità di effettuare il login tramite autenticazione HTTP *Basic*, oppure fornire una pagina apposita che presenti un form per l'inserimento delle credenziali[2]. Ogni operazione che scaturisce da una richiesta HTTP, inoltre, potrà accedere in ogni momento ai dettagli sull'utente che l'ha effettuata per autorizzarne o meno l'esecuzione.

4.1.10 Spring Data

Spring Framework, come già accennato, implementa nel suo *core* il concetto di accesso ai dati (tramite DAO, *Data Access Object*) e connessione al database (tramite JDBC). Inoltre mette a disposizione delle classi come `JdbcTemplate` e `RowMapper` che semplificano l'utilizzo di queste API gestendo autonomamente le operazioni più tediose.

Spring Data è un sotto progetto che espande queste caratteristiche, fornendo diverse tecniche per specificare i modelli e le relative repository (o DAO). Caratteristica molto interessante è la generazione automatica delle implementazioni di quest'ultime, rendendo necessario specificare solo l'interfaccia per esse: grazie ad uno schema preciso di nomenclatura per i metodi di accesso, il framework creerà *runtime* le istruzioni necessarie per interrogare la base di dati, avendo tutte le informazioni richieste specificate nei modelli (solitamente POJO[3] con annotazioni).

Viene così a mancare la necessità di dover scrivere codice ripetitivo e non legato alla pura logica di servizio.

Ovviamente è previsto un buon grado di flessibilità, permettendo di personalizzare il comportamento di ogni metodo dell'interfaccia, ma rendendola maggiormente dipendente dall'implementazione della base di dati scelta.

Creando uno strato di accesso ai dati definito da interfacce comuni (le *Repository*), è possibile slegare l'implementazione della persistenza (e le tecnologie usate per essa) dall'utilizzo dei dati contenuti al suo interno.

Per il progetto si è scelto di utilizzare *Spring Data JPA*, che permette di sfruttare, appunto, le *Java Persitance API*. Queste API permettono di definire entità e relazioni, tipiche di uno schema logico-relazionale, tramite l'utilizzo di semplici oggetti Java e delle apposite annotazioni: un'entità (o tabella del database) viene così descritta da un POJO, i cui attributi sono associati alle colonne e le chiavi esterne a riferimenti ad altri oggetti rappresentanti le opportune entità.

Lo standard JPA definisce solamente un set di interfacce, del quale esistono diverse implementazioni, sviluppate in progetti indipendenti. *Spring Data JPA* utilizza di default *Hibernate*, la più diffusa fra le alternative.

4.1.11 Spring Data Rest

Spring Data Rest è un progetto innovativo della famiglia *Spring Data*. Costruito su *Spring MVC*, permette di esportare automaticamente le *repository* specificate (di qualunque tipo siano, JPA,

[2] Questa funzionalità non è utilizzabile quando il server espone delle API REST, in quanto richiede la generazione dinamica di un documento HTML lato server.

[3] *Plain Old Java Object*, classi Java che non estendono nessun'altra classe esistente

MongoDB, Redis, ecc) come risorse *RESTful*. In altre parole, il componente si occupa di generare autonomamente dei controller MVC che implementino gli *endpoint* HTTP (indirizzi URL al quale il server risponde) necessari per eseguire tutte le operazioni esposte dalle *repository*.

Sfrutta inoltre *Spring HATEOAS*, che permette di creare rappresentazioni *REST* che seguono, appunto, i principi *HATEOAS* tramite l'implementazione di *HAL*, come decritto nei paragrafi precendenti.

4.1.12 AngularJS

Per la creazione delle interfacce client si è utilizzato *AngularJS*, un framework creato da Google per lo sviluppo di applicazioni JavaScript a pagine singola (*SPA, Single-page application*) completamente client-side, sfruttando gli standard a disposizione nei browser moderni: HTML5, JS e CSS3, principalmente. AngularJS implementa il modello *MVC* (*model-view-controller*) e *MVVM* (*model-view-viewmodel*)[4], avvicinandosi ai concetti propri dello sviluppo di UI desktop: è possibile quindi separare la logica del client dalla manipolazione del *DOM* (*Domain Object Model*) e dalla logica di visualizzazione.

Anche AngularJS, come Spring, sfrutta il meccanismo di *dependency injection*: le componenti riutilizzabili come i *model* (chiamati *provider*, o *service*) vengono inizializzate automaticamente e le loro istanze fornite ai moduli che ne richiedono l'utilizzo.

Per permettere questo isolamento fra le componenti di controllo (detti *controller* e scritti in JS, che manipolano i dati) e quelle di visualizzazione (quindi il codice HTML che compone la pagina), AngularJS espande il linguaggio di markup HTML fornendo delle direttive (*tag* e attributi) personalizzate, che permettono di specificare delle azioni proprie solo dell'interfaccia (ad es., il click su un link) direttamente nella vista, lasciando al *controller* il compito di definire la logica da eseguire. I dati generati dal controller vengono riportati nella vista tramite *data binding*.

Le componenti principali del framework sono quindi:

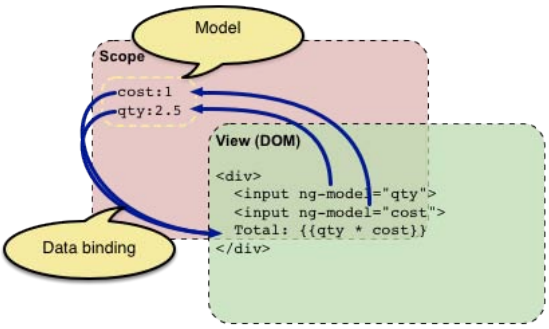
Service: definisce e modella i dati e le funzionalità condivise nella applicazione. Solitamente si occupa della comunicazione con il *backend*.

Controller: oggetto che racchiude parte della logica di funzionamento dell'applicazione. Collega i dati e le funzionalità offerte dei servizi con una vista, in entrambe le direzioni (fornisce le informazioni da mostrare all'utente ed elabora l'input da esso fornito).

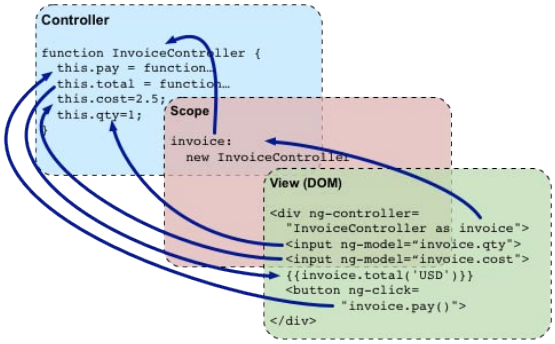
Scope: contenitore dei dati utilizzati da un *controller* ed accessibili dalla relativa *view*. Qualunque altro componente non può accederci, agevolando così la compartimentazione dell'applicazione.

Template: pagina o stralcio di codice HTML, esteso con markup apposito per accedere alle funzionalità e ai dati esposti dal *controller*.

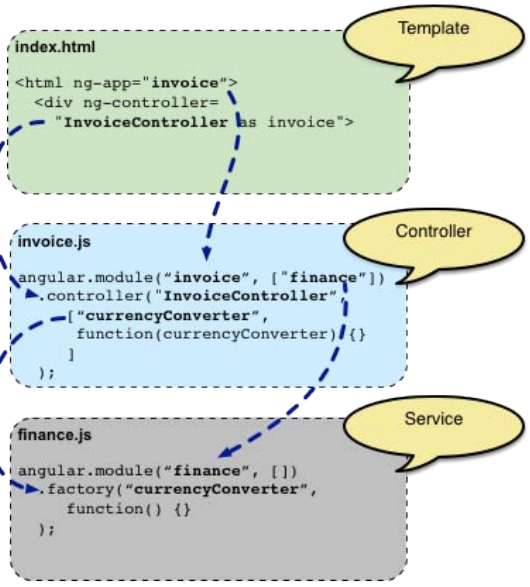
[4]Un pattern simile a MVC, dove il *Controller* è sostituito da un *ViewModel*, il cui scopo è effettuare il *binding* fra i dati del modello e quelli utilizzabili nella vista



(a) *Esempio di data binding fra view e scope. Quando cost e qty varieranno il DOM verrà aggiornato automaticamente. Viceversa, quando l'utente inserirà un valore nei tag input, questo verrà assegnato alla relativa variabile.*



(b) *Altro esempio di data binding con l'uso di un controller: insieme al binding bidirezionale, possibile con le variabili esposte come qty e cost, è possibile richiamare funzioni del controller. Queste verranno richiamate, e la view aggiornata, ogni qual volta il contenuto dello scope verrà modificato.*



(c) *Esempio di dependency injection e di interazione fra le componenti Template (vista), Controller (logica di controllo) e Service (modelli e funzionalità comuni)*

Figura 4.1: *I concetti alla base di AngularJS. (Fonte: https://docs.angularjs.org/guide/concepts)*

In Figura 4.1 sono riportati alcuni schemi esplicativi, estratti dalla documentazione ufficiale, che esemplificano graficamente i concetti qui esposti.

4.1.13 Peaks.js

Peaks.js è una libreria JavaScript sviluppata dalla *BBC* che permette la visualizzazione di un'onda sonora. Legandosi ad un player audio HTML5, può seguire la riproduzione e mostrare nel dettaglio la posizioni corrente nella tracciaf. Il componente è completamente *client side* e richiede dal server solo un file contenente i dati della forma d'onda da visualizzare. È possibile utilizzare due formati: binario, più compatto, e JSON, maggiormente compatibile con i diversi *browser*. Il primo infatti, richiede il supporto da parte di questi dei *Typed Arrays*, che permettono di manipolare buffer binari in JavaScript: ad oggi, le versioni aggiornate dei più famosi browser supportano questa feature.

Peaks.js permette inoltre di specificare e visualizzare, sovraimpressi all'onda, dei segmenti, aree evidenziate che coprono un determinato intervallo temporale e dei punti, eventi spot evidenziati con una linea ad un determinato *timestamp*. Entrambi questi tipi di *marker* possono essere modificabili, cioè l'utente può trascinarne gli estremi tramite il mouse all'istante d'interesse.

Altra funzionalità offerta è la possibilità di sfruttare le *Web audio API* per generare la *waveform* lato client direttamente dallo stream audio. L'interfaccia di accesso sviluppata in questa tesi non sfrutta questa funzionalità in quanto le tracce audio utilizzate hanno una durata che in media supera i 30 minuti, e la cui dimensione raggiunge i 70MB: elaborare lato client tali file su macchine di cui non si conoscono le capacità di calcolo potrebbe rendere inutilizzabile lo strumento per l'utente finale. I dati necessari per la forma d'onda vengono pertanto generati dal server durante la creazione della copia di accesso, sfruttando un apposito strumento complementare a *peaks.js*, chiamato *audiowaveform*, sviluppato sempre dalla *BBC*.

4.2 Panoramica del sistema

Il sistema sarà formato da una componente *backend* che esporrà delle API REST per la gestione dei dati e un *frontend* che fornirà l'interfaccia utente e gestirà le interazioni con esso. Tutti i dati verranno mantenuti in un database, accessibile solo dal *backend* e mai modificabile manualmente dagli utenti (tranne per l'amministratore del sistema). L'architettura scelta è quella di un'applicazione web, in modo da rendere più agevole l'utilizzo multiutente; i dati verranno caricati su un server, così da escludere la necessità di copiarli in ogni macchina che accede al software.

4.3 Implementazione della base di dati

A partire dallo schema ER presentato in 3.4.2, è stata implementata una nuova base di dati con il DBMS *PostgreSQL*, il cui schema logico-relazione è visibile in 4.2.

Per determinare gli attributi da assegnare a ciascuna tipologia di supporto è stata analizzata la base di dati esistente e verificato quanto estratto con gli operatori del laboratorio. Grazie al

loro contributo si è potuto modellare un insieme di proprietà e possibili loro valori completo ed esaustivo.

In Appendice B è riportato quanto raccolto; si può notare come la maggior parte degli attributi richieda un vincolo sui possibili valori inseribili: questi attributi sono stati implementati come nuovi tipi `Enum` in *PostgreSQL*, soddisfando così il vincolo richiesto senza la necessità di creare ulteriori tabelle.

Qualora si incontri, in futuro, un valore non presente tra quelli proposti, *PostgreSQL* permette di aggiungere nuove voci ad un tipo di dato *Enum* utilizzando i seguenti comandi SQL:

```
-- Aggiunge in coda alla lista
ALTER TYPE enum_t ADD VALUE 'new_value';
-- Aggiunge prima di existing_value
ALTER TYPE enum_t ADD VALUE 'new_value' BEFORE 'existing_value';
-- Aggiunge dopo di existing_value
ALTER TYPE enum_t ADD VALUE 'new_value' AFTER 'existing_value';
```

Nella tabella `Document` è stato inserito un attributo di testo denominato *additional_data*, in cui è permesso inserire stralci di dati come XML o JSON che il *backend* potrà utilizzare per gestire le personalizzazioni richieste dai committenti.

È stato ideato un metodo che sfrutta questo campo per permettere di specificare un valore inusuale per un attributo, qualora necessario, solo per un determinato documento: il software lo salverà in `additional_data`, nella notazione a lui più consona, in una voce la cui chiave corrisponde al nome dell'attributo. Quando, successivemente, dovrà leggere dalla base di dati i diversi attributi, interpreterà quanto inserito in `additional_data` e presenterà all'utente il valore in esso contenuto. In questo modo si potrà evitare l'aggiunta di un valore usato in casi rari o unici all'enumeratore definito nella base di dati, che diventerebbe di difficile consultazione.

I metadati dei file multimediali, che ne descrivono le caratteristiche tecniche e permettono di verificarne l'integrità nel tempo, saranno generati da strumenti esterni, analogamente a quanto avviene nel software esistente. Questi dati saranno salvati nella base di dati all'interno delle tabelle `Audio`, `Video` e `Photos` assieme ai riferimenti alla posizione dei file su disco. Provenendo da una sorgente esterna, si è deciso di non vincolare questi dati, utilizzando tipi base, come stringhe, interi e *float*.

4.4 Nuova interfaccia d'accesso

In parallelo alla progettazione e allo sviluppo della base di dati, è stata sviluppata l'interfaccia per l'accesso e l'analisi musicologica, che ha permesso di sperimentare tutte le tecnologie scelte, implementando tutte le richieste dettate dai requisiti. Si è pertanto deciso di espandere la base di dati già esistente con quanto necessario per gestire le analisi, ricalcando quanto proposto nel capitolo precedente. In questo modo quanto sviluppato potrà essere facilmente adattato alla nuova piattaforma, pur potendo sfruttare fin dal primo momento i dati già presenti, generati dagli operatori nello svolgimento dei loro compiti.

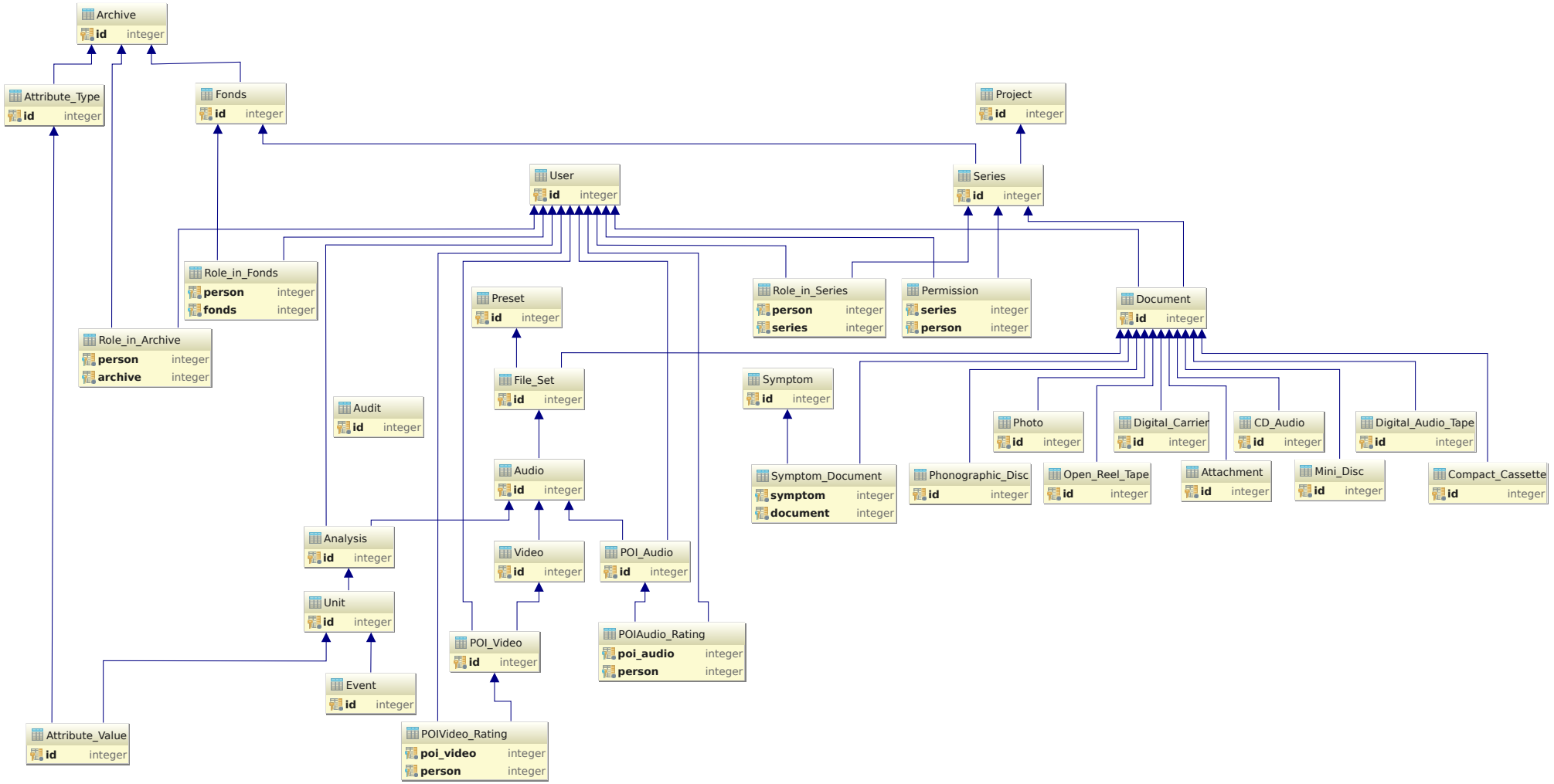


Figura 4.2: *Schema logico relazione della base di dati implementata*

4.4.1 Espansione della base di dati

Come anticipato nell'introduzione, il progetto utilizza la base di dati esistente, usata dal software *PsKit* e pertanto già popolata. La base di dati è stata così espansa per ospitare le tabelle necessarie all'analisi, ricalcando quanto proposto per la nuova versione in sviluppo. Così facendo si è agevolata la futura migrazione: le componenti software operanti nell'ambito dell'accesso troveranno inalterati i relativi concetti chiave e le loro relazioni.

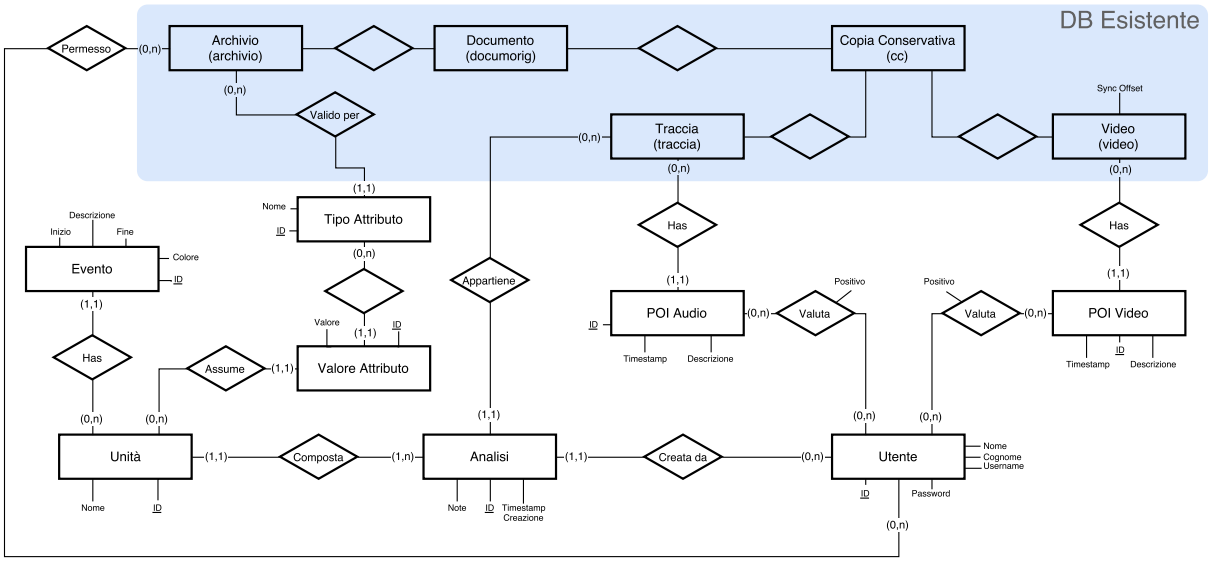


Figura 4.3: *Schema ER delle componenti d'accesso per la base di dati di PsKit*

In Figura 4.3 è riportato lo schema ER delle aggiunte apportate alla base di dati, assieme alle sole entità già esistenti che sono coinvolte in questa modifica (racchiuse nel riquadro). Le nuove entità seguono quanto descritto in 3.4.2; si può notare come lo schema pre-esistente, inoltre, esponga alcuni concetti in comune con quanto esposto nel precedente capitolo: parte dell'analisi dei requisiti si basa infatti sullo studio di questa base di dati. Esistono, però, alcune differenze:

- *Documento* è legato direttamente ad *Archivio*, mentre nel nuovo schema lo è a *Serie*.
- Il concetto di *Copia Conservativa* (unica per ogni *Documento*) è stato sostituito da quello *File Set*: in entrambi i casi, però, fungono da contenitori delle tracce multimediali
- *Traccia* e *Video* non sono legati in alcun modo. Da quanto emerso dallo studio del software *PsKit*, i nomi dei file vengono posti allo stesso valore, per cui l'associazione viene determinata dal confronto tra questi due attributi.

In Figura 4.4 è riportato, invece, lo schema logico-relazionale dell'espansione: si può notare come gli attributi appartenenti alle relazioni molti-a-molti siano stati riportate nelle relative tabelle, la cui chiave primaria è la combinazione delle due chiavi esterne verso le tabelle relative alle entità associate.

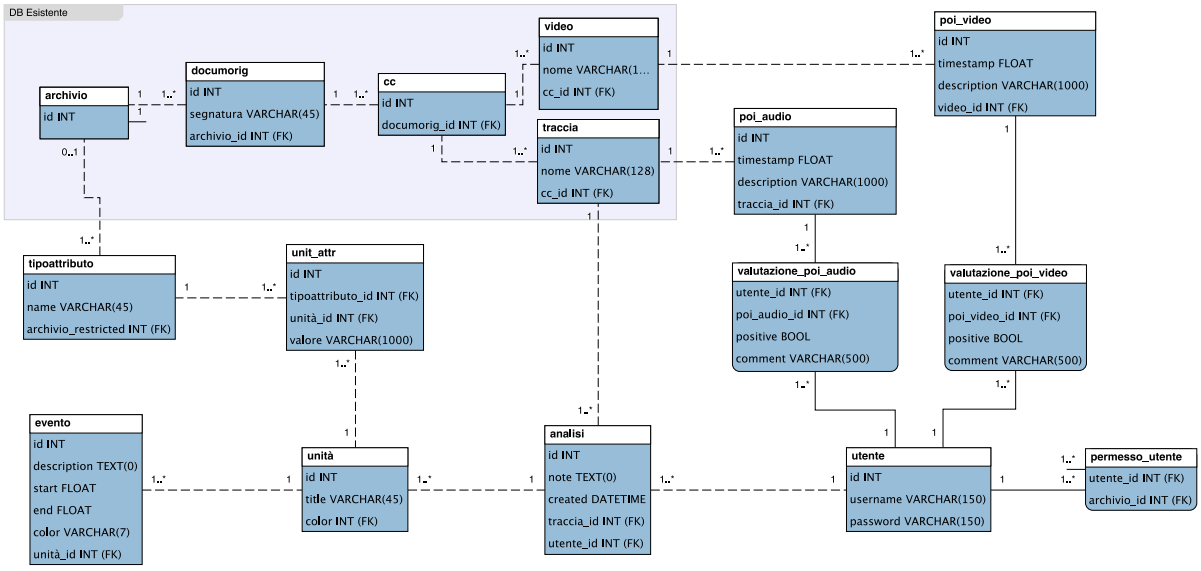


Figura 4.4: *Stesse modifiche dell'ER*

4.4.2 Componente server

Successivamente è stato sviluppato un server il cui scopo è fornire, accedendo alla base di dati, un servizio di API REST per accedere alle funzionalità esposte. Il software utilizza *Spring Boot* e, di conseguenza, i moduli *JPA*, *Spring Framework*, *Data*, *Data REST*, *Security* e *MVC*. Le componenti principali possono essere elencate come seguono:

Domain models Tramite l'uso di JPA sono state scritte le classi che modellano la struttura dei dati e le loro relazioni, così come specificate nella base di dati. Ogni classe corrisponde ad una tabella, i cui attributi sono mappati ai campi omonimi. Per specificare queste relazioni vengono utilizzate principalmente le seguenti annotazioni:

`@Entity`: assegna la classe ad una tabella specifica

`@Column`: mappa un campo della classe ad un attributo della tabella

`@OneToMany`, `@ManyToOne`, `@OneToOne`, `@ManyToMany`: specificano un'associazione definita nella base di dati, descrivendo la relazione tra una chiave esterna e la tabella di destinazione. In questo modo la relazione viene rappresentata come riferimento ad un'istanza della relativa classe Java (o una collezione di istanze, nel caso delle relazioni *a-molti*)

`@JoinColumn`, `@JoinTable`: accompagnano le annotazioni di associazione specificando come eseguire il *join* delle tabelle relazionate.

`@Id`: mappa un campo della classe alla chiave primaria della tabella. Nel caso di chiavi composite questo campo dovrà essere un istanza di una particolare classe creata *ad-hoc* contenente tutti i campi che le formano.

Grazie all'uso di questa metodologia, *Hibernate* può automaticamente costruire le query da inviare alla base di dati, per poi mappare i risultati in istanze di queste classi. Sarà così possibile manipolare i dati come puri oggetti Java, navigando tra le relazioni tramite i loro appositi metodi *getter* e *setter*.

Repository Ogni *domain model* rappresenta una riga di una tabella nella base di dati, ma non esegue direttamente operazioni di modifica su di essa: per creare, modificare, cancellare o semplicemente ottenere una lista filtrata di istanze di una certa entità è necessario definire una classe di servizio, che esponga le funzionalità richieste e ritorni le giuste istanze dei modelli. Questi servizi vengono astratti da *Spring Data* dall'interfaccia `Repository`: espandendola si possono definire i metodi per manipolare i relativi modelli secondo necessità. Definendo un'interfaccia non è necessario implementare questi metodi: sarà compito della libreria, in base al nome dato a ciascuno di essi, generare le istruzioni richieste per eseguire l'operazione. Il codice richiesto, infatti, non è legato alla logica operativa dell'applicazione, rivelandosi spesso ripetitivo, *boilerplate* e quindi facilmente generabile automaticamente per sollevare lo sviluppatore da questa necessità. I metodi più comuni definibili in queste interfacce sono:

`findOne()` : accetta una chiave primaria e ritorna la relativa istanza.

`findAll()` : ritorna tutte le righe disponibili. Se il tipo di ritorno estende `Page` e il metodo accetta come parametro un oggetto `Pageable`, l'output viene paginato secondo quanto indicato, in quanto una tabella potrebbe contenere un numero elevato di righe, non gestibili in un unico blocco.

`save()` : accetta un'istanza del modello per salvarne il contenuto nella base di dati, creando una nuova riga nella tabella, o aggiornandola se già presente.

`delete()` : accetta un'istanza del modello per rimuoverne la corrispettiva riga nella base di dati, se presente.

È possibile, inoltre, definire dei metodi di ricerca più mirati, specificando nel nome i criteri di filtraggio che andranno a comporre la clausola `WHERE` della query. Alcuni esempi di questo tipo di metodi dichiarati nel codice sono:

`findByUsername()` : trova tutte le istanze il cui attributo "username" equivale al parametro passato al metodo.

`findByArchiveAndFileSetCompleteTrue()` : ricerca le istanze di `Document` che appartengono ad un determinato archivio e che sono legate ad un `FileSet` il cui attributo booleano *complete* ha valore positivo. Si può notare come si possano attraversare più relazioni per definire query complesse.

Nel caso in cui fossero necessarie delle interrogazioni più complesse o non esprimibili come descritto, è possibile annotare il metodo con `@Query`, specificando un espressione SQL. È possibile utilizzare una query nativa, che verrà eseguita direttamente dalla base di

dati, oppure utilizzare JPQL (*Java Persistence Query Language*), che permette di definire delle interrogazioni sui modelli e i loro attributi. Quest'ultima opzione non espone nessun riferimento alle caratteristiche della base di dati a cui JPA si appoggia, rendendo il codice indipendente dalle caratteristiche peculiari del DBMS scelto.

Spring Data REST esporrà tutte le *repository* e i relativi metodi come risorse REST, lasciando allo sviluppatore il compito di configurare le policy di accesso, qualora necessarie.

Controller MVC Per espandere le funzionalità esposte da *Spring Data REST* e aggiungere ulteriori *endpoint* sono state implementate alcuni classi *controller*, i cui metodi, annotati con `@RequestMapping`, vengono eseguiti quando viene richiesto il relativo URL definito nell'annotazione. È stato reso possibile così accedere ai file multimediali collegati alle diverse copie conservative via HTTP, per permetterne la riproduzione remota nel client.

È stato possibile aggiungere, inoltre, gli *endpoint* creati fra i link ipermediali di ogni rappresentazione grazie all'implementazione di sottoclassi di `ResourceProcessor` per `Audio` (traccia e forma d'onda), `Video`, `Photo`, `VideoPoI` e `Document` (per accedere al report PDF). Allo stesso modo è stato aggiunto un *link* nei punti di interesse alla valutazione dell'utente corrente, se presente.

Configurazioni Tutte le componenti di *Spring Framework* possono essere personalizzate e configurate secondo le necessità dell'applicazione. Tutti le classi nel *namespace* che espongono l'annotazione `@Configuration` verranno scansionate all'avvio del programma per raccogliere i *beans* da loro creati: questi, che estendono una classe o implementano un'interfaccia specifica, vengono utilizzati dalle diverse componenti per definire il loro comportamento. La classe di configurazione `SecurityConfiguration`, ad esempio, che estende `WebSecurityConfigurerAdapter`, viene utilizzata da *Spring Security* per definire quale metodo di *hashing* usare per le password, quali sistemi di sicurezza aggiuntivi utilizzare e per quali URL richiedere l'autenticazione.

In Appendice C è riportato un esempio di codice Java per definire un *domain model* e una *repository*.

Nella gestione delle richieste HTTP per i file multimediali, per permettere al browser di saltare ad un istante arbitrario della traccia, si è reso necessario implementare la tecnica denominata *byte serving*: il client richiede, tramite l'*header* `Range`, di cominciare (o riprendere) il download da un preciso offset, espresso in byte: il server risponderà con il codice `206 Partial Content`, specificando nell'*header* di risposta `Content-Range` l'intervallo di dati che invierà, procedendo successivamente all'invio dello stream di byte relativo.

4.4.3 Sicurezza - gestione dell'autenticazione e autorizzazione

La sicurezza del sistema è implementata grazie agli strumenti messi a disposizione di *Spring Security*. Il primo passo è consisito nella creazione della classe `AuthorizedUser`, che implementa l'interfaccia `UserDetails`, con cui la libreria rappresenta gli utenti nel sistema per conoscere i loro dettagli, tra cui nome utente e password. Successivamente è stata implementata

l'interfaccia `UserDetailsService` con la classe `JpaUserDetailsService`, il cui scopo è fornire a *Spring Security* le istruzioni necessarie per creare un'istanza di `UserDetail` a partire da un *username* fornito durante il processo di autenticazione: da come si può intuire dal nome dell'implementazione, la classe utilizzerà i dati degli utenti contenuti nella base di dati, sfruttando i modelli e i servizi definiti in precendenza.

Spring Security si occuperà successivamente di richiedere le credenziali all'utente, ottenere la relativa istanza di `UserDetail` (se presente) e confrontane la password con quanto comunicato dal client. Qualora l'utente venga autenticato correttamente, la libreria si prende carico di mantenerne la sessione inviando autonomamente al client il relativo *cookie* ad ogni richiesta ricevuta.

Secondo quanto definito nella configurazione, verrà permesso all'utente di accedere agli URL delle API REST. Per definire nel dettaglio a quali operazione di ogni risorsa l'utente può accedere, sono state aggiunte le annotazioni `@PreAuthorize` e `@PostAuthorize` ai metodi delle *repository* che, valutando l'espressione *SpEL*[5] specificata, autorizzano rispettivamente l'esecuzione e l'invio al client del risultato. L'eventuale esito negativo della verifica verrà notificato al client impostando il codice di stato HTTP `401 Unauthorized`.

L'implementazione di queste verifiche ha permesso di limitare l'accesso alle sole analisi e relativi componenti creati dall'utente che le richiede, sia in lettura che scrittura.

4.4.4 Componente client - interfaccia web

Per accedere ai servizi offerti dalla componente *backend* è stata sviluppata un'interfaccia web che implementa quanto richiesto dai requisiti per l'analisi musicologica. Allo scopo di rendere completamente indipendente la parte client dal server, questa interfaccia è stata implementata come applicazione web *single-page* con *Angular JS*: tutta la logica relativa alla comunicazione con l'utente e presentazione dei dati è quindi affidata al codice *JavaScript*, che verrà eseguito nel *browser*. Il server viene così sollevato dal compito di generare ogni genere di *vista*, affidando ad esso la sola gestione dei dati e rendendo possibile, di conseguenza, espletare tutte le funzionalità tramite le API REST definite.

Essendo un'applicazione web, l'interfaccia definisce un unica pagina, `index.html`, il cui compito è indicare quali *script* e *stylesheet* caricare, oltre che definire il contenitore principale nel quale visualizzare le componenti della UI. I file `app.js` e `app.config.js` contengono il modulo denominato `app` che definisce i diversi stati dell'applicazione e configura i servizi forniti al caricamento della pagina. Come descritto, l'interfaccia impiega una singola pagina HTML, per cui ogni cambiamento della vista non determina un ricaricamento dell'intero documento con relativo cambio di URL; pertanto ogni "pagina" dell'applicazione viene definita da uno *stato*, mentre le transizioni tra questi delineano il *workflow* dell'interfaccia.

Grazie alla modularità propria di *AngularJS*, è possibile utilizzare alcuni componenti sviluppati da terzi per integrare le funzionalità della propria applicazione: tutti i *widget* e gli stili che compongono l'interfaccia, infatti, sono forniti dalla libreria *Angular Material*, estensione

[5] Un semplice linguaggio per definire brevi espressioni, formattare stringhe di testo e manipolare oggetti Java

ufficiale del framework che si pone l'obbiettivo di implementare il *Material Design*[6] attraverso direttive (elementi HTML personalizzati che definiscono parti dell'interfaccia riutilizzabili) e servizi di *Angular*. Queste dipendenze sono gestite tramite *Bower*, un *package manager* studiato appositamente per le applicazioni web, estremamente facile da usare: in questo modo è possibile scaricarle al bisogno senza dover tenere traccia delle loro versioni manualmente[7]. Il file `bower.json` raccoglie tutti i pacchetti necessari per il funzionamento dell'applicazione e che, essendo sorgenti JavaScript a loro volta, vengono inclusi in `index.html`.

Alcun moduli, chiamati *service*, definiscono dei servizi comuni all'intera applicazione e indipendenti dallo stato e dalla vista corrente: il loro compito è fornire un'interfaccia uniforme ai dati o alle funzionalità condivise. Tra questi spicca `Backend`, che espone alcuni metodi per richiedere i dati dal server, appoggiandosi alla libreria *angular-hypermedia*[8] per la gestione delle risorse JSON HAL.

Tra gli stati principali figura la pagina di *login*: qualora una richiesta al server generi un errore di accesso non autorizzato si viene reindirizzati a questo *form*, dove sono richiesti nome utente e password da inviare al *backend* tramite il servizio `Authentication`. Nel caso in cui l'autenticazione vada a buon fine, il servizio salva i dettagli dell'utente per renderli disponibili all'intera applicazione.

La vista principale dell'interfaccia, sul quale si è concentrato maggiormente il lavoro svolto, è `analysis`: come dice il nome, fornisce l'interfaccia e la logica per gestire un'analisi, in cui vengono caricate tutte le informazioni relative al documento scelto. La riproduzione viene affidata un elemento `audio` HTML5: il codice si occupa di sincronizzare la riproduzione di quest'ultimo con il riproduttore video, se presente. La vista si compone di più aree, così definite:

Waveform: Viene mostrata la forma d'onda della traccia, sia nella sua interezza (riga inferiore), che nel dettaglio correntemente selezionato (riga superiore). Vengono qui rappresentati i punti di interesse come linee e gli eventi come segmenti colorati, modificabili trascinandone le estremità. Cliccando su un punto qualsiasi la riproduzione della traccia verrà portata all'istante selezionato.

Video: se previsto dalla copia conservativa, viene visualizzata la documentazione video registrata durante il riversamento. Cliccando sull'area di riproduzione il video viene espanso al di sotto della forma d'onda (sostituendosi al resto dell'interfaccia), per permettere di cogliere più agevolmente i dettagli del supporto originale (Figura 4.6a).

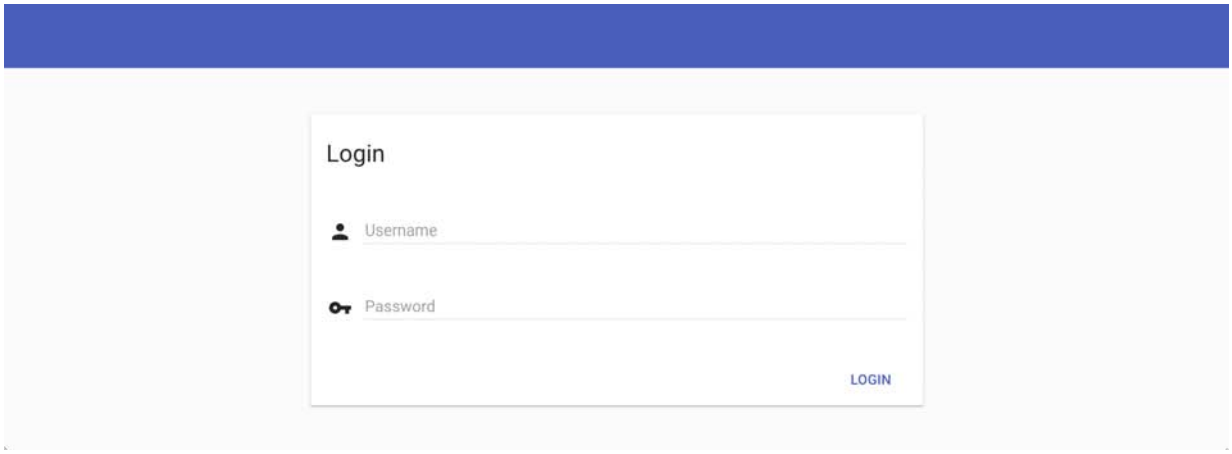
Lista delle unità: Elenco delle unità presenti nell'analisi, da cui è possibile eliminarle o crearne di nuove. Selezionandone una si aprirà la lista di attributi ed eventi.

Lista di attributi: mostra nel dettaglio gli attributi dell'unità corrente. È possibile crearne di nuovi ed eliminare o modificare quelli esistenti nel tipo e nel valore. I tipi di attributo

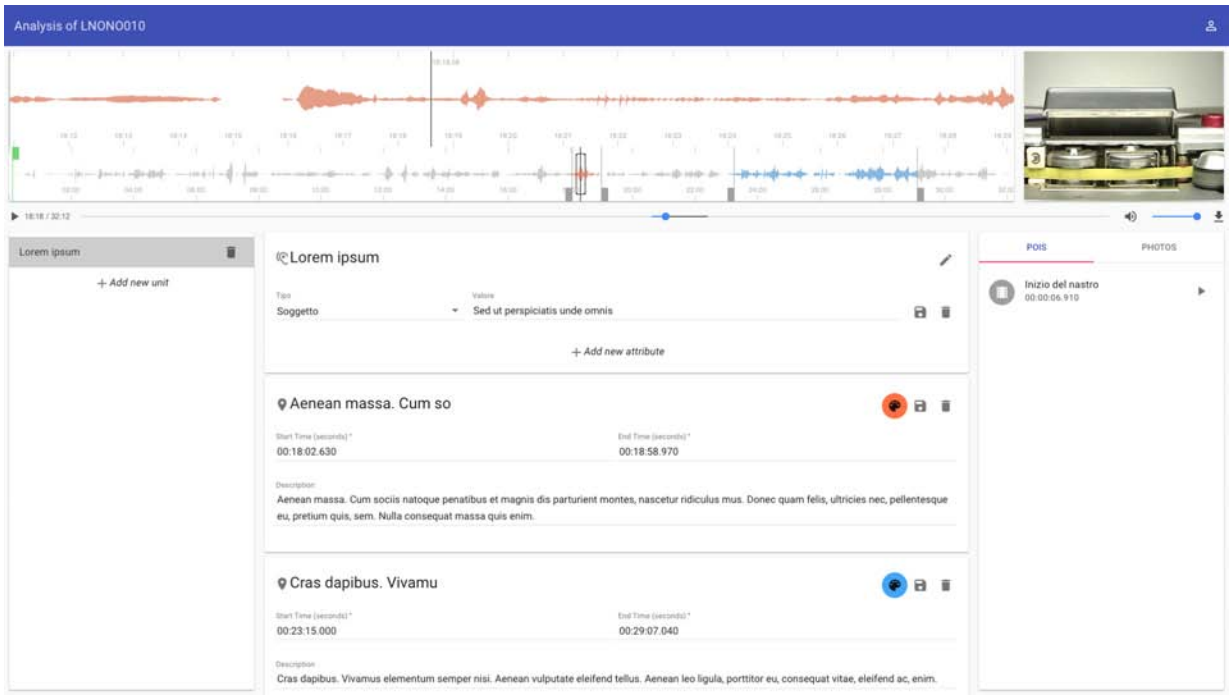
[6] Linguaggio di design definito da Google ed implementato in buona parte dei suoi prodotti, primo fra tutti Android (dalla versione 5.0)

[7] L'uso di un dependency manager, inoltre, elimina la necessità di includere il codice di terzi nella repository del progetto, mantenendola snella e focalizzata sulle funzionalità sviluppate autonomamente.

[8] `https://github.com/jcassee/angular-hypermedia`



(a) *Login*



(b) *Analisi*

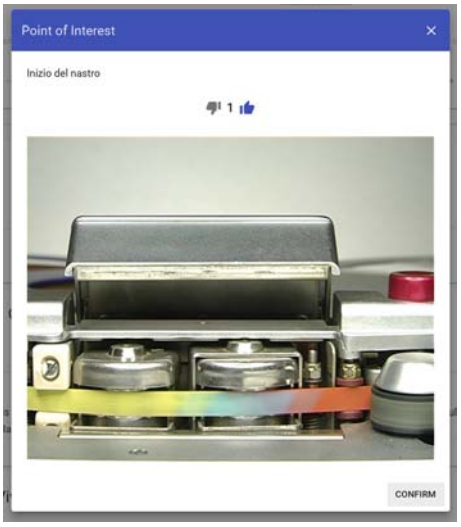
Figura 4.5: *Schermate principali dell'interfaccia d'accesso.*

selezionabili sono limitati a quelli associati all'archivio a cui appartiene il documento, così come definito dalla base di dati.

Lista di eventi: ogni evento viene visualizzato nel suo riquadro (chiamato, in *Material Design*, *card*), dove è possibile modificarne le informazioni relative e selezionare un colore per identificarlo nella forma d'onda. In coda alla lista è presente un pulsante per l'aggiunta di un nuovo evento.

(a) *Riproduttore video ingrandito per una migliore visibilità*

(b) *Scheda delle foto*



(c) *Finestra di dialogo per un Punto di Interesse*

Figura 4.6: *Dettagli dell'interfaccia di analisi*

Punti di interesse: la scheda mostra un elenco dei punti di interesse, sia audio, sia video, da cui è possibile, tramite l'apposito pulsante, posizionare il *player* all'istante definito. Cliccando sulla voce, invece, si aprirà il relativo *dialog* di dettaglio.

Foto: posizionato come scheda insieme ai punti di interesse, mostra un elenco di foto associate al documento (Figura 4.6b). Cliccando su ogni anteprima si aprirà una scheda del *browser* con l'immagine a risoluzione piena.

Dettaglio di un Punto di Interesse: mostra la descrizione completa di un punto d'interesse e

l'eventuale frame del video associato all'instante temporale definito (Figura 4.6c). Sono presenti, inoltre, i controlli per la valutazione del punto di interesse, raffigurati come due pollici in direzioni opposte; al centro viene riportato il bilancio totale delle valutazioni ricevute da tutti gli utenti.

Gli istanti temporali sono gestiti nel codice (sia server che client) come numero decimale di secondi trascorsi, per semplificare le operazioni di manipolazione e confronto. È stata quindi implementata un'estensione di un elemento di `input` testuale HTML, al quale, grazie alle funzionalità del framework, sono stati aggiunti un *parser* che interpreta il tempo sessagesimale fornito dall'utente ed un *formatter* che esegue l'operazione contraria: in questo modo la variabile associata all'input (tramite il *two-way binding*) sarà sempre in formato decimale, ma l'utente potrà visualizzarla e modificarla nella sua rappresentazione più naturale.

4.4.5 *Deploy*: metodi e destinazione finale

Per rendere operative entrambe le componenti client e server, è necessario che esse siano installate su una macchina diversa da quella di sviluppo, senza servizi accessori e dove non siano presenti tutti gli strumenti di modifica e compilazione del codice; non sarà necessario, infatti, effettuare *debug* o modifiche del software durante la sua esecuzione. In questo modo si riducono, inoltre, i possibili punti di attacco alla macchina (qualora questa sia esposta alla rete pubblica).

Il *backend*, quindi, deve essere compilato per produrre i binari da poter eseguire. Il progetto utilizza un sistema di *project management* denominato *Maven* e sviluppato dalla *Apache Foundation*, i cui compiti sono, previa definizione di un apposito file, denominato `pom.xml`:

- Definire e gestire le dipendenze di un progetto, scaricandole da una *repository* centralizzata.
- Compilazione dei sorgenti e preparazione all'esecuzione tramite organizzazione corretta dei file prodotti
- Creazione degli artefatti Java richiesti, come un pacchetto JAR eseguibile.
- Esecuzione di operazioni personalizzate tramite l'espansione delle funzionalità con appositi *plugin*, gestiti similmente alle dipendenze.
- Gestione dei file di risorse aggiuntivi da includere nel pacchetto compilato

Nel progetto, *Maven* è stato configurato per fornire le giuste dipendenze dai componenti *Spring*, così come consigliato dalla loro documentazione. La compilazione del codice non ha richiesto nessuna personalizzazione particolare: grazie a *Spring Boot* è possibile infatti integrare nel pacchetto JAR un'istanza di *Apache Tomcat*[9] e di istruzioni di *bootstrap* atte a configurarlo per eseguire il codice compilato. Secondo la documentazione ufficiale, questo sistema può essere

[9]Tomcat è un *application server* per l'esecuzione di *servlet* sviluppate in Java. È sviluppato e mantenuto dalla *Apache Software Foundation*

utilizzato anche in produzione (cioè in sede finale di utilizzo del software), poiché l'istanza di *Tomcat* utilizzata è esattamente equivalente ad una installata e configurata manualmente nella macchina.

Il *fontend*, invece, utilizzando linguaggi di *scripting*, non va compilato, ma necessita di alcune ottimizzazioni ed elaborazioni per poter ottenere i risultati migliori nella sua esecuzione. Il progetto del client, infatti, utilizza oltre 20 file fra JavaScript, HTML e CSS, senza contare le dipendenze esterne: tutti questi andranno trasferiti al browser dell'utente e per ognuno sarà necessario effettuare una richiesta HTTP; il traffico generato risulterà così enorme e per nulla ottimizzato.

La soluzione sviluppata prevede di raccogliere tutti questi file ed unirli in un unico documento per linguaggio, ottenendo un singolo file HTML, un JavaScript e un foglio CSS. Opzionalmente, questi file possono essere minimizzati[10] per occupare il minimo spazio e, di conseguenza, banda in download possibile. Uno script appositamente creato esegue queste operazioni, generando un pacchetto inferiore ai 2 MB a fronte degli oltre 20 del progetto non elaborato.

Il progetto *Spring* sviluppato può essere configurato per servire anche dei file statici, che andranno inclusi come risorse nel pacchetto generato in compilazione. Per agevolare le operazioni di *deploying* (cioè di messa in opera dell'applicazione) si è deciso di sfruttare questa caratteristica per unire i file generati da client e server in un unico pacchetto eseguibile: il server, oltre ad esporre le API sviluppate, fornirà al browser i file necessari ad eseguire l'interfaccia client. Per automatizzare l'intero processo di compilazione e creazione del pacchetto finale, *Maven* è stato configurato per eseguire, grazie ad un apposito plugin, lo script di ottimizzazione del *frontend* descritto sopra, includendo successivamente i file generati alla giusta posizione nel pacchetto JAR.

Eseguendo, quindi, il singolo comando `mvn package` verranno eseguiti i passi appena descritti per generare il pacchetto eseguibile da trasferire alla macchina di destinazione. Altri servizi, come il DBMS utilizzato, andranno invece configurati separatamente. Una volta reso operativo il server, sarà possibile accedere all'interfaccia collegandosi al suo indirizzo (alla porta 8080, se non configurato diversamente) da un qualsiasi browser.

[10] eliminazione o riduzione degli *whitespace* dal codice per ridurre le dimensioni di un file di scripting. Alcuni metodi sostituiscono anche i nomi di variabili e funzioni per ottenere un ulteriore guadagno. Ovviamente questi sistemi rendono il codice difficilmente leggibile.

Capitolo 5

Conclusioni e sviluppi futuri

I progressi tecnologici e l'avvento dell'era digitale hanno portato enormi progressi nell'ambito del restauro e della conservazione dei documenti sonori storici. La ricerca ha così potuto definire delle metodologie sistematiche standard per conformare le operazioni di rimediazione e generare efficientemente copie conservative affidabili e, soprattutto, complete. Software appositamente sviluppati per questi scopi possono offrire un valido supporto agli operatori in ogni passo delle procedure, agevolando le operazioni più ripetitive e inclini ad errori.

In questo ambito si pone il metodo adottato dal Laboratorio di Sonologia Computazionale dell'Università di Padova, descritto in [3], che ha portato allo sviluppo del software denominato *PsKit*, correntemente utilizzato. È stata quindi svolta un'analisi tecnica dell'applicazione e delle sue componenti, per portare alla luce le cause di alcune problematiche riscontrate dagli operatori.

Dall'interazione con essi, insieme ai componenti del team del laboratorio, è stata proposta un'analisi dei requisiti per una nuova piattaforma che superi le limitazioni incontrate, rendendo ancora più efficace l'applicazione della metodologia di conservazione. Oltre a ciò, la piattaforma si pone come mezzo per riunire diversi progetti sviluppati in laboratorio, per permettere una migliore integrazione tra di essi e, soprattutto, garantire l'accesso condiviso ai dati delle copie conservative e d'accesso, centralizzando la loro gestione.

Successivamente all'analisi, dai risultati raccolti è stato sviluppato un modello dei dati per lo stoccaggio delle informazioni generate in maniera strutturata. Sono state proposte due versioni del modello, dove la prima è stata scartata poiché incline a problematiche legate all'inconsistenza dei dati, dovute al suo approccio estremamente dinamico e alla elevata interdipendenza con il software a livello superiore; la seconda variante è stata sviluppata ponendo come obiettivo l'autonomia nella gestione dei dati per renderli accessibili e consistenti, indipendentemente dal software che ne farà uso. Dal modello si è quindi potuto implementare la base di dati tramite l'uso di *PostgreSQL*.

L'architettura della nuova piattaforma è stata definita come un server centrale che possa fornire quanto contenuto nella base di dati, tramite API REST su protocollo HTTP, a uno o più client, sviluppati con tecnologie web. Implementando tutte le funzionalità richieste attraverso questa interfaccia è possibile isolare la base di dati, il cui accesso sarà mediato esclusivamente dal software di *backend*, centralizzando la gestione dell'intero patrimonio informativo delle opere conservate.

Nel contempo è stato affrontato un secondo aspetto nella definizione della nuova piattaforma: l'accesso e l'analisi musicologica. Presso il CSC si è già affrontata questa tematica, raccogliendo i requisiti e le opinioni dai potenziali utilizzatori finali di tale strumento, potendo così definire nel modello dei dati le componenti necessarie ad ospitarne i contenuti. La nuova interfaccia è stata quindi equipaggiata per poter sfruttare e supportare quanto già definito dai precedenti lavori e di fatto si pone come un buon use case per valutare la solidità delle tecnologie proposte.

Lo strumento è stato sviluppato, quindi, ricalcando quanto proposto per l'architettura della nuova piattaforma: un server, scritto in *Java*, che espone delle API REST ad un client web, sviluppato in HTML e JS con l'ausilio di *AngularJS*, accessibile da remoto tramite un comune *browser web*; sono state implementate, inoltre, soluzioni per l'autenticazione e l'autorizzazione dell'accesso alle risorse fornite dal server.

Il progetto, di cui sono state poste le basi, dovrà essere sviluppato ulteriormente. Sarà necessario lo sviluppo del software *backend* che implementi i vincoli non soddisfabili dalla base di dati e i servizi necessari all'espletamento delle funzionalità richieste; sarà possibile attingere dall'esperienza maturata nello sviluppo del server per le funzionalità di accesso: molte soluzioni alle problematiche incontrate potranno essere applicate al nuovo software. Allo stesso modo, la componente *frontend*, l'interfaccia attraverso la quale gli utenti accederanno ai dati e i servizi, potrà sfruttare quanto già sviluppato per l'accesso e l'analisi che, inoltre, potrà essere facilmente adattata al nuovo *backend*, in quanto appositamente progettata e sviluppata in quest'ottica.

Successivamente, i dati e metadati correntemente presenti nella base di dati dovranno essere migrati nella nuova piattaforma in maniera automatica, data l'elevata quantità di documenti in essa conservati.

In futuro sarà possibile, inoltre, integrare altri progetti sviluppati all'interno del laboratorio e riunire così tutte le ricerche portate avanti nell'ambito della conservazione e dell'accesso alle opere sonore. Alcune applicazioni che potranno essere sviluppate o integrate sono:

- Un software di intelligenza artificiale per l'estrazione automatica dei punti di interesse dalla registrazione video di un nastro [9].
- Un'applicazione Android per la fruizione di documenti sonori su nastro che simuli l'esperienza d'uso e di ascolto del magnetofono fisico (progetto REMIND [6]), sfruttando le copie di accesso fornite dalla piattaforma.
- Strumenti per la creazione di report per le copie conservative, modellati sulle richieste dei committenti, che uniscano i metadati raccolti nella base di dati in un formato stampabile personalizzabile. Una possibile implementazione potrebbe sfruttare un template *LaTeX* automaticamente popolato e compilato in un documento PDF.

Appendici

Appendice A

Vincoli per gli attributi dei documenti in PsKit

Come specificato nella documentazione del progetto *Gra.Fo*, precursore del progetto *PsKit*, gli attributi per ogni documento sono raccolti nella tabella `documorig`, indipendentemente dal tipo di supporto. Per questo, non tutti sono sempre applicabili o richiedono di essere specificati dall'utente (poiché costanti o facilmente ricostruibili): la seguente tabella descrive quindi la natura di ciascun attributo per ogni tipo di supporto.

Ogni cella riporta uno dei seguenti valori:

✓ : è richiesto un input dall'utente

× : l'attributo non è richiesto e andrà impostato a `NULL`

k : l'attributo viene impostato automaticamente dal software con un testo predefinito

f : l'attributo è noto a priori o funzione del valore di altri attributi, quindi ricavabile senza richiederlo all'utente.

	Bobina	**Cassetta**	**CDA**	**DAT**	**Supporto digitale**	**Disco Fono-grafico**
velocita	✓	f	×	×	×	✓
riavvolgimentoprima	✓	✓	×	✓	×	×
riavvolgimentodopo	f	f	×	f	×	×
testonote	k	k	k	k	k	k
allegati	k	k	k	k	k	k
tecnicaregistrazione	f	f	f	f	f	✓
registrazionead	f	f	f	f	f	f
larghezzanastro	f	f	×	f	×	×
diametro	✓	×	✓	×	×	✓
custodiatipo	✓	✓	✓	✓	×	✓

custodiamarca	✓	✓	✓	✓	×	✓
supportomarca	✓	✓	✓	✓	×	✓
supportomodello	✓	✓	✓	✓	×	✓
flangiamarca	✓	×	×	×	×	×
flangiamateriale	✓	×	×	×	×	×
proceduraarchiv	*f*	*f*	*f*	*f*	*f*	*f*
statoconservazione	✓	✓	✓	✓	✓	✓
supportohdd	×	×	×	×	✓	×
eq	✓	✓	×	×	×	✓
fc	×	×	*f*	✓	✓	×
nr	✓	✓	×	×	×	×
codifica	×	×	*f*	×	✓	×
bitdepth	×	×	*f*	✓	✓	×
bitrate	×	×	×	×	✓	×
formato	×	×	*f*	×	✓	×
numerolati	✓	*f*	*f*	*f*	×	✓
numeropiste	✓	*f*	*f*	*f*	✓	✓
segnale	✓	✓	✓	✓	✓	✓
numerotracce	×	×	✓	✓	✓	×
noteconservazione	✓	✓	✓	✓	✓	✓
matrice	×	×	×	×	×	✓
numerotake	×	×	×	×	×	✓
grooveorientation	×	×	×	×	×	✓
layersurfacematerial	*f*	*f*	✓	*f*	×	✓
layersubstratematerial	*f*	*f*	✓	*f*	×	✓

Appendice B

Tipologie di supporti e attributi

Tabella B.1: *Attributi comuni*

Attributo	Tipo	Valori	
original_signature	String		
completed	Boolean		
conservation_status	Enum	Excellent, Good, Fair, Very Bad	
case_material	Enum	Metal, Paper, Plastic, Missing, Other	
case_brand	Enum	Agfa, Audiotape, Basf, Fuji, RCA, ...	
carrier_brand	Enum	Agfa, Audiotape, Basf, Fuji, RCA, ...	
carrier_type	Enum	Open reel tape, Compact cassette, DAT, CD, ...	
carrier_model	String		

Tabella B.2: *Cassette*

Attributo	Tipo	Default	Valori
is_micro	Boolean	False	
minutes	Enum		60, 90, 120, ...
tracks_per_side	Enum		1, 2, 4, 8, 16, 24, Unknown
previously_storage **storage_after_treatment**	Enum Enum		Beginning side A, Beginning side B, Other
layer_substr_material	Enum		PE/PET/Mylar, PE/PVC based, Unknown
speed	Enum		1 7/8 ips, 3 3/4 ips, 7 1/2 ips, . . .
noise_red	Enum		CONTROLLARE
equalization	Enum		CONTROLLARE
recording_technique	Constant	Magnetic	
recording_ad	Constant	Analogic	
tape_width	Constant	0.15 in (3.81 mm)	

Tabella B.3: *Bobine*

Attributo	Tipo	Default	Valori
flange_brand	Enum		Agfa, Ampex, RCA, ReVox, ...
sides_count	Integer		1, 2
pancake	Boolean		
hub	Enum		CINE', 'NAB', 'AEG DIN'
tracks_per_side	Enum		1, 2, 4, 8, 16, 24, Unknown
tracks_config	Enum		Mono, Stereo, Other
previously_storage **storage_after_treatment**	Enum Enum		Heads-out, Tails-out, 2-sided tape, Other
layer_substr_material	Enum		Cellulose acetate, PE/PET/Mylar, ...
layer_surface_material	Enum		Cellulose acetate, Unknown, Other
flange_diameter	Enum		3 1/4 in, 4 in, 4 1/4 in, 5 in, ...
diameter_core	Decimal		
tape_width	Enum		2 in, 1 in, 1/2 in, 1/4 in
speed	Enum		1 7/8 ips, 3 3/4 ips, 7 1/2 ips, ...
noise_red	Enum		CONTROLLARE Dolby A, Dolby B, Dolby C, Dolby S, ...
equalization	Enum		Ampex, DIN, DIN Type I, CCIR, NAB
recording_technique	Constant	Magnetic	
recording_ad	Constant	Analogic	

Tabella B.4: *Digital Audio Tape*

Attributo	Tipo	Default	Valori
tracks_count	Integer		
previously_storage **storage_after_treatment**	Enum Enum		Heads-out, Tails-out, Other
sample_rate	Enum		32 kHz, 44.1 kHz, 48 kHz, 96 kHz, 98 kHz
layer_substr_material	Enum		PE/PET/Mylar, PE/PVC based, Unknown
bit_depth	Enum		12 bit, 16 bit, 24 bit
sides_count	Constant	1	
recording_technique	Constant	Magnetic	
recording_ad	Constant	Digital	
tape_width	Constant	0.2 in (4mm)	

Tabella B.5: *Dischi Fonografici*

Attributo	Tipo	Default	Valori
tracks_count	Integer		
sides_count	Integer		
groove_orientation	Enum		45/45, Horizontal, Vertical
layer_substr_material	Enum		Glass, Paper, Plastic material, Unknown, Other
layer_surface_material	Enum		Bachelite, Shellac, Vinyl, Unknown, Other
diameter	Enum		7 in, 10 in, 12 in
recording_technique	Enum		Electro-Mechanical, Magnetical, Mechanical
speed	Enum		16 rpm, 33 1/3 rpm, 45 rpm, 78 rpm
equalization	Enum		CONTROLLARE
recording_ad	Constant	Analogic	

Tabella B.6: *CD Audio*

Attributo	Tipo	Default	Valori
tracks_count	Integer		
format	Enum		Standard, Mini CD, ”Business-card” size
layer_surface_material	Enum		Alluminium, Gold, Silver, Unknown, Other
layer_substr_material	Constant	Polycarbonate	
recording_technique	Constant	Optical	
recording_ad	Constant	Digital	
sample_rate	Constant	44,1 kHz	
bit_depth	Constant	16 bit	
encoding	Constant	PCM	

Tabella B.7: *MiniDisc*

Attributo	Tipo	Default	Valori
tracks_count	Integer		
format	Enum		MD, MD-HiMD, Hi-MD
rec_mode	Enum		stereo SP, mono SP, LP2, LP4, Hi-LP, ...
bit_rate	Enum		256 Kbps, 292 Kbps, 146 Kbps, 132 Kbps, ...
codec	Enum		ATRAC, ATRAC3, ATRAC3plus, PCM
recording_technique	Constant	Optical	
recording_ad	Constant	Digital	

Tabella B.8: *Altro Supporto Digitale*

Attributo	**Tipo**	**Default**	**Valori**
capacity	Integer		
recording_technique	Constant	Magnetical	
recording_ad	Constant	Digital	

Appendice C

Esempio di *domain model* e *repository* nel backend

```
@Entity(name = "analisi")
public class Analysis {
  // ------------------------------------------------------
  //region Primary key

  @Id @GeneratedValue
  private int id;

  //endregion
  // ------------------------------------------------------
  //region Attributes

  @Column(columnDefinition = "TEXT") private String note;

  //endregion
  // ------------------------------------------------------
  //region Foreign Keys

  @ManyToOne
  @JoinColumn(name="utente_id", referencedColumnName="id", nullable=false)
  private User user;

  @ManyToOne
  @JoinColumn(name="traccia_id", referencedColumnName="id", nullable=false)
  private Track track;

  //endregion
  // ------------------------------------------------------
  //region Reverse mappings

  @OneToMany(mappedBy="analysis")
  private Collection<Unit> units;
```

```
  //endregion
  // ------------------------------------------------------

  // Getters and setters not shown here
}
```

```
@PreAuthorize("isAuthenticated()")
public interface AnalysisRepository extends Repository<Analysis, Integer> {
  // Return ONE by id only if it belongs to the user that requested it
  // null checking is used to let pass every request for non-existing data.
  // This way it will generate a 404 http error instead of a 403 (not authorized)
  @PostAuthorize("returnObject == null or returnObject?.user?.id == principal.id")
  Analysis findOne(Integer id);

  // Return ALL the items that belongs to the user that requested it
  @Query("select e from #{#entityName} e where e.user.id = ?#{ principal.id }")
  Iterable<Analysis> findAll();

  @PostAuthorize("@analysisRepository.findOne(#id)?.user?.id == principal.id")
  boolean exists(@P("id") Integer id);

  // Save only if owned by the current user
  @PreAuthorize("#a?.user?.id == principal.id")
  Analysis save(@P("a") Analysis analysis);

  // Delete only if owned by the current user
  @PreAuthorize("@analysisRepository.findOne(#id)?.user?.id == principal.id")
  void delete(@P("id") Integer id);

  // Delete only if owned by the current user
  @PreAuthorize("#a?.user?.id == principal.id")
  void delete(@P("a") Analysis analysis);
}
```

Appendice D

Standard di equalizzazione e velocità nei supporti analogici

Di seguito vengono riportate le equalizzazioni tipiche per le audiocassette, in combinazione con le velocità di scorrimento del nastro.

Velocità	**Sigla**	**Anno**	**Cost. bassi**	**Cost. alti**
30 ips, 76 cm/s	IEC 2 AES	1981, current standard	∞	17.5 µs
30 ips, 76 cm/s	CCIR IECI DIN	1953-1966 1968 1962	∞	35 µs
15 ips, 38 cm/s	IEC 1 CCIR DIN BS	1968, current standard 1953 1962	∞	35 µs
15 ips, 38 cm/s	NAB EIA	1953, current standard 1963	3180 µs	50 µs
7 ½ ips, 19 cm/s	IEC 1 DIN (studio) CCIR	1968, current standard 1965 1966	∞	70 s
7 ½ ips, 19 cm/s	IEC 2 NAB DIN (home) EIA RIAA	1965, current standard 1966 1963 1968	3180 µs	50 µs
7 ½ ips, 19 cm/s	Ampex (home) EIA (proposed)	1967	∞	50 µs

7 ½ ips, 19 cm/s	CCIR IEC DIN BS	fino al 1966 fino al 1968 fino al 1965	∞	100 µs
3 ¼ ips, 9.5 cm/s	IEC 2 NAB RIAA	1968, current standard 1965 1965	3180 µs	90 µs
3 ¼ ips, 9.5 cm/s	DIN	1962	3180 µs	120 µs
3 ¼ ips, 9.5 cm/s	DIN	1955-1961	∞	200 µs
3 ¼ ips, 9.5 cm/s	Ampex (home) EIA (proposed)	1967	∞	100 µs
1 ⅞ ips, 4.75 cm/s	IEC DIN	1971, current standard 1971	3180 µs	120 µs
1 ⅞ ips, 4.75 cm/s	IEC DIN RIAA	1968-1971 1966-1971 1968	1590 µs	120 µs
1 ⅞ ips, 4.75 cm/s	IEC Type I	1974, current standard	3180 µs	120 µs
1 ⅞ ips, 4.75 cm/s	DIN Type I	1968-1974	1590 µs	120 µs
1 ⅞ ips, 4.75 cm/s	Type II and IV	1970, current standard	3180 µs	70 µs

Bibliografia

[1] *TC-05: Handling and Storage of Audio and Video Carriers*. IASA Technical Commitee, 2014.

[2] George Boston. *Safeguarding the documentary heritage: a guide to standards, recommended practices and reference literature related to the preservation of documents of all kinds*. United Nations Educational, Scientific and Cultural Organization, 1998.

[3] Federica Bressan and Sergio Canazza. A systemic approach to the preservation of audio documents: Methodology and software tools. *Journal of Electrical and Computer Engineering*, 2013:5, 2013.

[4] Mike Casey and Bruce Gordon. Sound directions: Best practices for audio preservation. Technical report, Harvard University, Cambridge, Mass, USA; Indiana University, Bloomington, Ind, USA, 2007.

[5] Elizabeth Cohen. Preservation of audio in folk heritage collections in crisis. *Proceedings of Council on Library and Information Resources*, 2001.

[6] Daniele Colanardi. Progetto e realizzazione di un'interfaccia utente scheumorfica per la fruizione di documenti sonori storici su dispositivi mobile. Bachelor's thesis, University of Padova, 2014.

[7] Ray Edmondson. *Memory of the World: general guidelines to safeguard documentary heritage*. Information Society Division, United Nations Educational, Scientific and Cultural Organization, 2002.

[8] Aleksandra Mežnarić Karafin. Digitization of sound recordings as an example for preservation of oral and music folklore heritage. In *Proceedings of the 1st International Conference on The Future of Information Sciences (INFuture '07): 'Digital Information and Heritage'*, pages 139–152, 2007.

[9] Mirco Maniero. Analisi automatica di nastri magnetici con reti neurali e tecniche di visione computazionale. Master's thesis, University of Padova, 2016.

[10] John McIlwaine, Jean-Marc Comment ICA, Clemens de Wolf, Dale Peters, Borje Justrell ICA, Marie-Thérèse Varlamoff, and Sjoerd Koopman. Guidelines for digitization projects

for collections and holdings in the public domain, particularly those held by libraries and archives, 2002.

[11] Mary Miliano. *The IASA cataloguing rules: a manual for the description of sound recordings and related audiovisual media*. International Association of Sound and Audiovisual Archives, 1999.

[12] Christopher Ann Paton. Preservation re-recording of audio recordings in archives: Problems, priorities, technologies, and recommendations. *The American Archivist*, 61:188–219, 1998.

[13] Dietrich Schüller. The ethics of preservation, restoration, and re-issues of historical sound recordings. *Journal of the audio engineering society*, 39(12):1014–1017, 1991.

[14] Abby Smith. Why digitize? *Microform & imaging review*, 28(4):110–119, 1999.

[15] William Storm. The establishment of international re-recording standards. *Phonographic Bulletin*, 27:5–12, 1980.

[16] Sergio Canazza Valentina Burini, Federico Altieri. Rilevamenti sperimentali per la conservazione attiva dei documenti sonori su nastro magnetico: individuazione delle curve di equalizzazione. In *Extending interactivity. Atti del XXI CIM - Colloquio di Informatica Musicale*, 2016.

[17] ZKM. The digital oblivion: substance and ethics in the conservation of computer-based art. In *Proceedings of the International Symposium of the Research Project Digital Art Conservation*. Zentrum für Kunst und Medientechnologie Karlsruhe (ZKM), 2010.