Scuola universitaria professionale della Svizzera italiana
Dipartimento tecnologie innovative

SUPSI

Applicazione Desktop per la riproduzione di musica: MPFree

Studente/i
Marco Lorusso, Giacomo Zecirovic

Relatore
Amos Brocco

Correlatore
Edoardo Terzi

Corso di laurea
Ingegneria Informatica

Anno
2024/2025

Data 27/05/2025

Indice generale

Capitolo 1

1.1 Introduzione

La progettazione di applicazioni software nasce spesso dall'osservazione di bisogni concreti e dalla volontà di proporre soluzioni pratiche a problematiche reali. In quest'ottica, il presente progetto si sviluppa a partire da un'analisi del contesto d'uso e delle criticità riscontrate dagli utenti in ambito multimediale, per poi definire in modo chiaro gli obiettivi da perseguire durante lo sviluppo.

I prossimi paragrafi illustrano, rispettivamente, il problema affrontato e le finalità del progetto, delineando il percorso seguito per arrivare alla realizzazione di un'applicazione funzionale, usabile e orientata all'esperienza dell'utente.

1.1.1 Contesto e problema

Nel mondo digitale odierno, l'ascolto della musica rappresenta una delle attività quotidiane più diffuse, tanto sul web quanto tramite dispositivi locali. Nonostante l'ampia offerta di servizi di streaming, esiste ancora una forte esigenza di applicazioni desktop leggere, personalizzabili e offline per la riproduzione di musica in formato locale.

Molti utenti desiderano una soluzione semplice, priva di pubblicità e che permetta la gestione diretta dei propri file audio e playlist personali. Tuttavia, le applicazioni esistenti spesso presentano interfacce complesse o carenze nella gestione delle playlist e dei metadati, oppure richiedono risorse eccessive per funzionare correttamente su macchine meno recenti.

Il progetto risponde a questa esigenza proponendo lo sviluppo di un'applicazione desktop user-friendly per la riproduzione musicale, che consenta all'utente di interagire in modo immediato con i propri brani preferiti.

1.1.2 Obiettivi del progetto

Lo scopo del progetto è sviluppare un music player desktop in JavaFX, in grado di offrire una riproduzione fluida e intuitiva di file audio (formato MP3), accompagnata da funzionalità essenziali ma ben progettate. Gli obiettivi principali includono:

1. **Analisi dei requisiti**: Definire le funzionalità chiave e i requisiti tecnici, sia funzionali che non funzionali.
2. **Scelta delle tecnologie**: Utilizzare JavaFX per l'interfaccia grafica e JLayer per la gestione dei file audio.
3. **Progettazione dell'interfaccia**: Creare un design chiaro e coerente, seguendo le linee guida di usabilità.
4. **Implementazione incrementale**: Costruire l'applicazione in modo modulare, iniziando dalle funzionalità di base (riproduzione, stop, pausa, skip) fino alla gestione avanzata delle playlist.

5. **Testing accurato**: Verificare il corretto funzionamento del software tramite test unitari e di sistema.
6. **Documentazione**: Redigere una documentazione dettagliata sullo sviluppo e fornire una guida utente accessibile.

Inoltre, l'app mira a fornire un’esperienza personalizzabile, con una visualizzazione dei metadati del brano (artista, titolo, durata) e la possibilità, se desiderato, di integrare API esterne come Last.fm per il recupero automatico delle informazioni musicali.

Capitolo 2

2.1 Requisiti

Per garantire uno sviluppo efficace e orientato all'utente, è fondamentale definire in modo chiaro ciò che l'applicazione deve fare e come deve comportarsi. In questa fase, l'attenzione si concentra sull'identificazione dei requisiti, distinti in funzionali e non funzionali, che rappresentano rispettivamente le funzionalità operative del software e le caratteristiche qualitative che ne influenzano l'affidabilità, l'usabilità e le prestazioni.

Le sezioni seguenti descrivono nel dettaglio le capacità che l'applicazione deve offrire agli utenti, insieme agli standard qualitativi che ne guidano la progettazione e lo sviluppo.

2.1.1 Requisiti funzionali

I requisiti funzionali definiscono le funzionalità principali che l'applicazione musicale deve offrire agli utenti per essere considerata completa ed efficace. Di seguito è riportato l'elenco dettagliato delle funzionalità implementate o in corso di sviluppo:

- **Caricamento di file MP3**
  Permette agli utenti di selezionare e caricare brani audio in formato .mp3 dal file system locale.
- **Riproduzione del brano selezionato**
  Avvia la riproduzione audio del file scelto.
- **Pausa e ripresa della riproduzione**
  Consente di mettere in pausa e successivamente riprendere la riproduzione del brano.
- **Arresto della riproduzione**
  Interrompe completamente l'ascolto del brano.
- **Navigazione tra le tracce**
  Fornisce controlli per passare al brano successivo o precedente all'interno della playlist.
- **Riproduzione continua senza interruzioni (Seamless Playback)**
  Garantisce la continuità dell'ascolto tra i brani, senza pause percettibili.
- **Creazione di playlist personalizzate**
  Consente all'utente di creare nuove playlist contenenti una selezione di brani.
- **Gestione delle tracce nella playlist**
  Aggiunta, rimozione e modifica dell'ordine dei brani in una playlist.
- **Salvataggio delle playlist**
  Permette di salvare una playlist su disco per un utilizzo futuro.
- **Caricamento di playlist salvate**
  Consente di importare una playlist precedentemente salvata.

- **Riordinamento delle playlist**
  L'utente può modificare manualmente l'ordine delle tracce all'interno di una playlist.
- **Controllo del volume**
  Offre un controllo a scorrimento per alzare o abbassare il volume della riproduzione.
- **Funzionalità Mute/Unmute**
  Consente di silenziare o riattivare l'audio con un solo clic.
- **Visualizzazione del tempo di riproduzione**
  Mostra il tempo trascorso e il tempo totale del brano in riproduzione.
- **Ricerca all'interno del brano (Seek)**
  L'utente può saltare a un punto specifico del brano attraverso una barra di scorrimento temporale.
- **Visualizzazione delle informazioni del brano**
  Mostra metadati come titolo, artista e durata, se disponibili.
- **Ripresa automatica dell'ultimo brano ascoltato**
  All'avvio dell'applicazione, riprende automaticamente la riproduzione dell'ultimo brano ascoltato.
- **Interfaccia utente user-friendly**
  L'app presenta un design semplice e intuitivo, con layout chiari e accessibili anche a utenti non esperti.

2.1.2 Requisiti non funzionali

I requisiti non funzionali descrivono le caratteristiche qualitative che il sistema deve rispettare per garantire un'esperienza utente adeguata, affidabilità tecnica e manutenibilità:

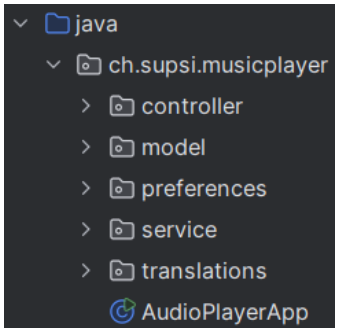
- **Usabilità**
  L'interfaccia utente deve essere intuitiva, minimale e conforme alle linee guida HIG (Human Interface Guidelines), facilitando l'accesso alle funzionalità principali con pochi clic.
- **Prestazioni**
  Il sistema deve garantire tempi di risposta rapidi per ogni azione (riproduzione, pausa, caricamento brani), mantenendo l'uso della CPU e della RAM entro limiti accettabili anche su hardware di fascia media.
- **Compatibilità**
  L'applicazione deve *funzionare* su diversi sistemi operativi desktop compatibili con Java (Windows, Linux, macOS).
- **Affidabilità**
  Il sistema deve essere robusto contro input non validi o file corrotti, notificando l'utente con messaggi di errore chiari.
- **Manutenibilità**
  Il codice sorgente deve essere ben strutturato e documentato per facilitare modifiche e aggiornamenti futuri.

- **Sicurezza**
  L’applicazione non deve accedere o modificare file di sistema se non esplicitamente richiesto dall’utente. Non deve raccogliere dati personali o inviare informazioni online senza consenso.

Capitolo 3

3.1 Architettura del sistema

Il progetto MPFree segue un'architettura a strati (layered architecture) con una chiara separazione delle responsabilità:

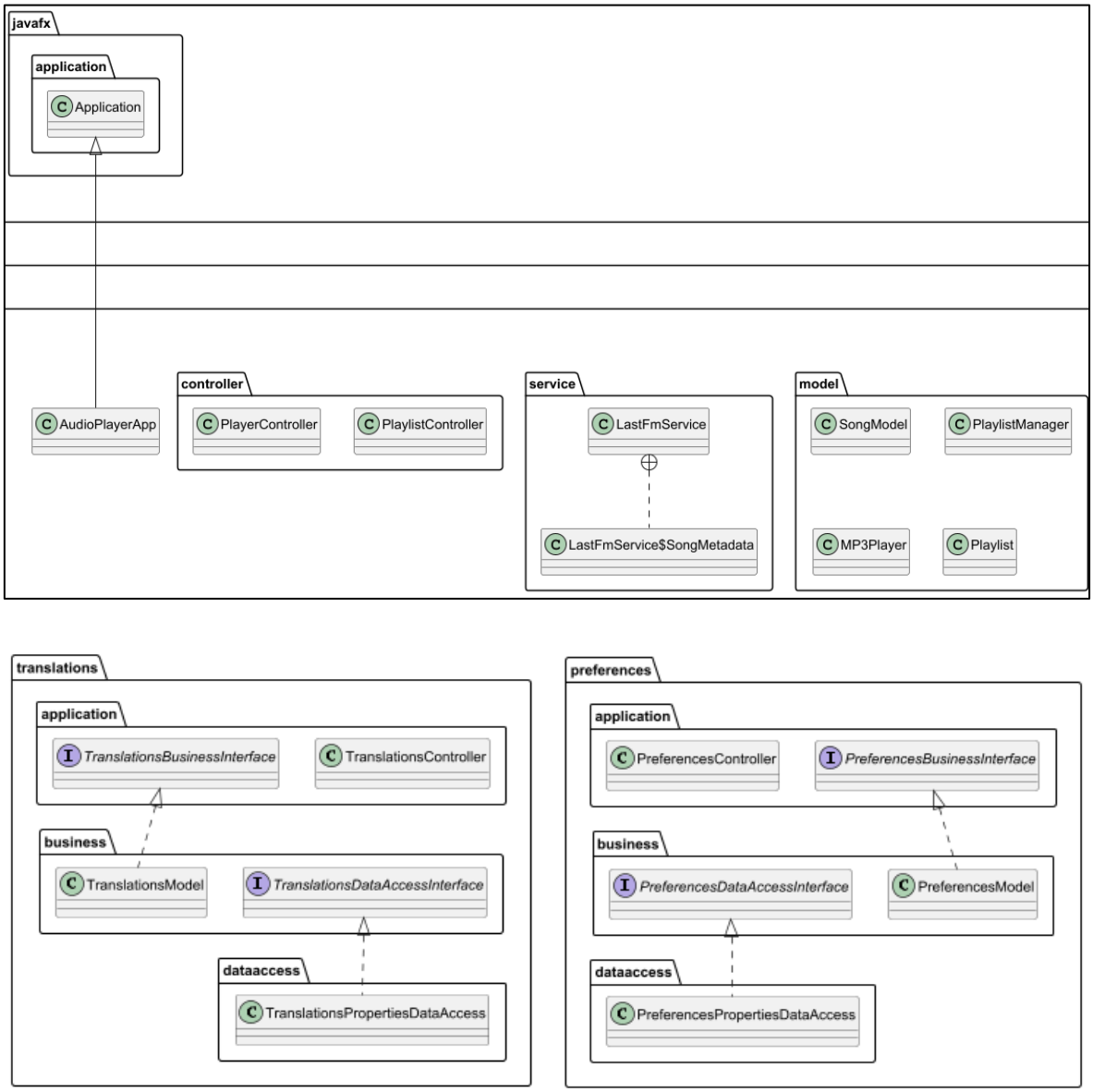


Architettura del progetto

Qui di seguito i componenti di questa architettura:

- **controller**: Contiene i controller dell'interfaccia utente, come PlayerController (per la riproduzione) e PlaylistController (per la gestione delle playlist). Questi controller interagiscono con i layer sottostanti.
- **service**: Dedicato ai servizi esterni e di utilità, come LastFmService per interagire con il servizio Last.fm e la sua classe interna.
- **model**: Rappresenta i dati e la logica di business principale, con classi come SongModel (per le informazioni sulle canzoni), PlaylistManager (per gestire le playlist), MP3Player (la logica di riproduzione audio) e Playlist (la struttura dati di una singola playlist).
- **preferences**: Gestisce le preferenze dell'applicazione, strutturato su più livelli (applicazione, business, data access) con interfacce e implementazioni concrete.
- **translations**: Gestisce le traduzioni, anch'esso organizzato in layer simili al package preferences (applicazione, business, data access) con le rispettive interfacce, controller e modelli. Questo indica una chiara separazione delle responsabilità per la gestione delle lingue.

3.2 Diagramma di classi



Il diagramma delle classi illustra l'architettura del Music Player, organizzata in package distinti per separare le responsabilità. Si evidenziano i package per l'interfaccia utente e i controller, la logica di business e i modelli dati, i servizi esterni, e la gestione delle preferenze e delle traduzioni, quest'ultime implementate con un'architettura a strati che include interfacce di business e di accesso dati. Le connessioni tra le classi e le interfacce indicano dipendenze e relazioni di implementazione, riflettendo un design modulare e stratificato.

3.3 Tecnologie Utilizzate

3.3.1 JavaFX

JavaFX è un framework moderno per lo sviluppo di applicazioni desktop ricche di interfaccia grafica. A differenza di Swing, suo predecessore, JavaFX offre:

- Un sistema di layout più flessibile e moderno
- Supporto nativo per CSS
- Animazioni fluide
- Gestione avanzata degli eventi
- Supporto per FXML per la separazione tra UI e logica

Implementazione nel Progetto:

- Utilizzo di FXML per definire le interfacce utente
- Styling tramite CSS per un'interfaccia moderna e responsive
- Implementazione di controlli personalizzati per la riproduzione audio
- Gestione degli eventi per i controlli di riproduzione
- Animazioni fluide per le transizioni tra le schermate

3.3.2 JLayer

JLayer è una libreria Java specializzata nella riproduzione di file MP3. Le sue caratteristiche principali includono:

- Decodifica efficiente dei file MP3
- Supporto per lo streaming audio
- Gestione avanzata del buffer
- Controllo preciso della riproduzione

Implementazione nel Progetto:

- Integrazione con il sistema di riproduzione audio
- Gestione efficiente della memoria per file di grandi dimensioni
- Implementazione di controlli di riproduzione avanzati
- Gestione del buffer per una riproduzione fluida

3.3.3 Maven

Maven è un tool di automazione build che gestisce:

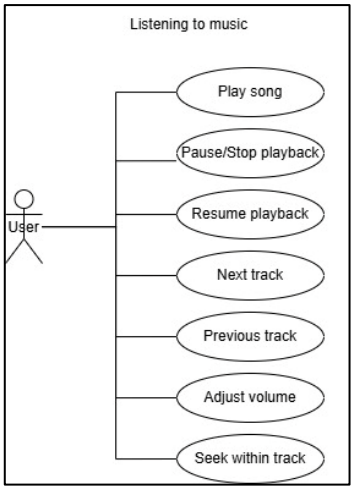
- Dipendenze del progetto
- Ciclo di vita della build
- Testing
- Deployment

3.4 Use Case

3.4.1 Riproduzione musicale

Questo diagramma rappresenta le funzionalità principali offerte all'utente per la gestione della riproduzione musicale.

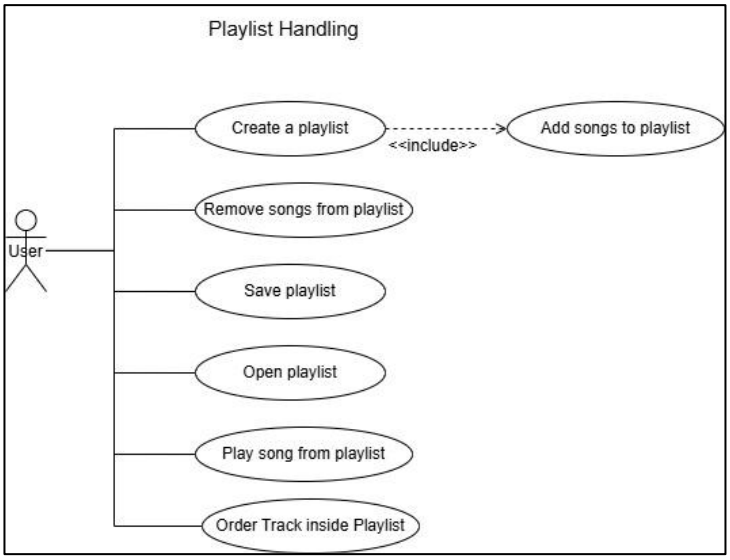
Ogni caso d'uso è stato progettato per riflettere un'intenzione chiara dell'utente e favorire un'esperienza utente semplice e diretta. Non sono presenti relazioni improprie tra i casi d'uso: le azioni sono indipendenti e accessibili individualmente in qualsiasi momento durante l'esecuzione dell'applicazione.



Use case: Listening to music 1

3.4.2 Gestione delle playlist

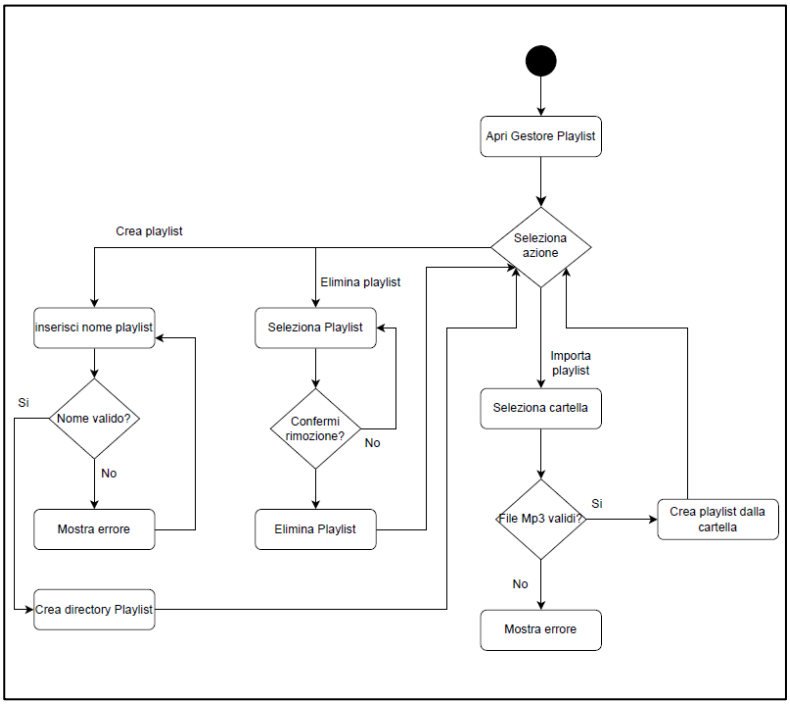
In questo diagramma sono descritte le azioni disponibili all'utente per creare e gestire le proprie playlist musicali.

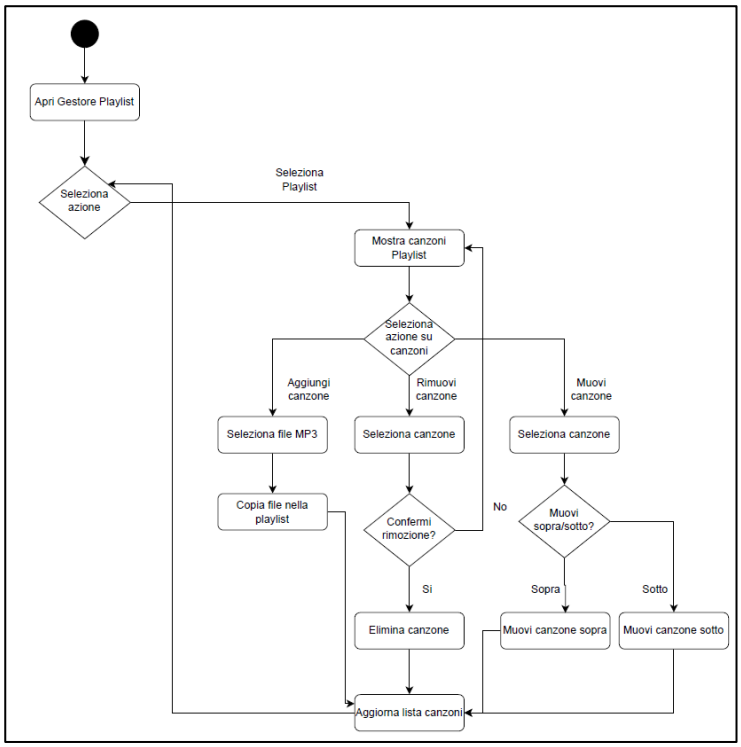


Use case: Playlist Handling

È stato utilizzato il meccanismo UML <<include>> per rappresentare l'inclusione obbligatoria dell'azione di aggiunta di brani all'interno della creazione della playlist. Questo riflette un comportamento ripetuto e necessario per dare significato alla playlist stessa.

3.5 Diagramma di attività – Gestione Playlist





Il diagramma illustra il flusso di gestione delle playlist nel lettore musicale. Partendo dall'apertura del Playlist Manager, l'utente può creare nuove playlist, importare cartelle di musica, eliminare playlist esistenti e gestire le canzoni all'interno di esse. Per ogni azione, il sistema verifica la validità delle operazioni e richiede conferma quando necessario.

Le operazioni principali includono l'aggiunta e rimozione di canzoni, lo spostamento delle tracce nella playlist e la riproduzione della playlist selezionata. Il diagramma mostra come il sistema mantenga la coerenza dei dati attraverso controlli di validazione e aggiornamenti automatici delle liste dopo ogni modifica.

Capitolo 4

4.1 Implementazione

Dopo aver definito i requisiti e progettato la struttura generale dell'applicazione, si passa alla fase di implementazione, durante la quale le funzionalità previste prendono forma attraverso l'uso di tecnologie e paradigmi software appropriati. In questa sezione vengono illustrate le principali scelte architetturali adottate, i componenti chiave dell'applicazione e le modalità con cui sono stati realizzati i meccanismi fondamentali, come la riproduzione audio e la gestione delle playlist.

L'obiettivo è fornire una panoramica chiara e organizzata delle soluzioni tecniche implementate, evidenziando l'approccio modulare seguito e la logica alla base delle principali funzionalità del sistema.

4.1.1 Architettura software: pattern MVC

L'applicazione è stata progettata secondo il pattern architetturale MVC (Model–View–Controller), per garantire modularità, manutenibilità e separazione delle responsabilità:

- **Model**: rappresenta lo stato dell'applicazione, comprese le classi relative a file audio, playlist e metadati. Gestisce le operazioni di caricamento, salvataggio e aggiornamento dei dati.
- **View**: costituita da componenti JavaFX, è responsabile della rappresentazione grafica. L'interfaccia utente è stata progettata per essere chiara, responsiva e user-friendly, secondo le HIG (Human Interface Guidelines).
- **Controller**: coordina l'interazione tra Model e View. Gestisce gli eventi generati dall'utente (ad esempio, la pressione dei pulsanti di controllo audio), aggiorna lo stato del Model e riflette i cambiamenti nella View.

Questo approccio ha facilitato lo sviluppo incrementale del progetto, permettendo al team di lavorare in parallelo sui vari componenti e rendendo agevole l'aggiunta di nuove funzionalità.

4.1.2 Sistema di Riproduzione Audio

Il sistema di riproduzione audio è implementato utilizzando la libreria JLayer per la decodifica MP3. La classe MP3Player gestisce:

1. Caricamento dei file audio:

```
public void load(File file) {
    stop();

    try {
        this.currentFile = file;
        SongModel song = new SongModel(file);
        Platform.runLater(() -> currentTrack.set(song));

        // Calcola la durata effettiva
        double durationInSeconds = calculateDuration(file);

        Platform.runLater(() -> {
            totalDuration.set(durationInSeconds);
            totalTimeString.set(formatTime(durationInSeconds));
        });
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Il caricamento di un file audio rappresenta il primo passo fondamentale per la riproduzione di un brano.

Il metodo load(File file) si occupa di interrompere qualsiasi riproduzione in corso e di preparare il nuovo file selezionato.

All'interno del blocco try, il file viene associato all'istanza corrente, viene creato un nuovo oggetto SongModel che rappresenta il brano e, tramite Platform.runLater, si aggiorna la traccia corrente nell'interfaccia utente. Successivamente, viene calcolata la durata effettiva del brano tramite il metodo calculateDuration, e anche questi dati vengono propagati all'interfaccia utente. In caso di errori, viene stampato lo stack trace per facilitare il debug.

Questo approccio garantisce che ogni volta che l'utente seleziona un nuovo file, tutte le informazioni e lo stato dell'applicazione vengano aggiornati in modo coerente e reattivo.

2. Controllo della riproduzione:

```
public void play() {
    if (currentFile == null) return;

    if (isPaused) {
        resume();
        return;
    }

    stopPlayback();
    playFileFromBeginning();
}
```

Il controllo della riproduzione è gestito dal metodo play(), che si occupa di avviare o riprendere la riproduzione del brano selezionato. Se non è stato caricato alcun file, il metodo termina immediatamente. Se la riproduzione era stata precedentemente messa in pausa, viene semplicemente ripresa dal punto in cui era stata interrotta. In caso contrario, viene fermata qualsiasi riproduzione in corso e si avvia la riproduzione del file dall'inizio. Questo meccanismo assicura che la riproduzione sia sempre sincronizzata con lo stato dell'applicazione e che l'utente possa gestire facilmente sia la ripresa che l'avvio di nuovi brani.

3. Gestione del volume:

```
public void setVolume(double value) {
    volume = Math.max(0.0, Math.min(1.0, value));

    for (FloatControl control : volumeControls) {
        if (control.getType() == FloatControl.Type.VOLUME) {
            control.setValue((float) volume);
        } else if (control.getType() == FloatControl.Type.MASTER_GAIN) {
            float min = control.getMinimum();
            float max = control.getMaximum();
            float range = max - min;
            control.setValue(min + (float) (volume * range));
        }
    }
}
```

La gestione del volume è implementata nel metodo setVolume, che consente di impostare il livello del volume in modo preciso e sicuro. Il valore del volume viene limitato tra 0.0 e 1.0 per evitare errori. Successivamente, il metodo itera su tutti i controlli di volume disponibili (ad esempio, per diversi dispositivi audio) e aggiorna il valore effettivo del volume. Se il controllo supporta il tipo VOLUME, il valore viene impostato direttamente; se invece supporta solo MASTER_GAIN, il valore viene calcolato in base al range consentito dal dispositivo. Questo garantisce una compatibilità estesa con diversi sistemi audio e una regolazione fine del volume percepito dall'utente.

4.1.3 Sistema di Gestione delle Playlist

Il sistema di gestione delle playlist implementa:

1. Caricamento delle playlist:

```
public void loadPlaylist(Playlist playlist) {
    this.currentPlaylist = playlist;
    currentPlaylistProperty.set(playlist);
    currentPlaylistIndex = -1;

    if (playlist != null && !playlist.getSongs().isEmpty()) {
        playTrackAtIndex(0);
    }
}
```

Il caricamento di una playlist è gestito dal metodo loadPlaylist. Questo metodo aggiorna la playlist corrente dell'applicazione, ne aggiorna la proprietà osservabile (utile per il binding con l'interfaccia utente) e reimposta l'indice del brano corrente. Se la playlist non è vuota, viene automaticamente avviata la riproduzione del primo brano. In questo modo, l'utente può passare rapidamente da una playlist all'altra e iniziare subito l'ascolto senza ulteriori interazioni.

2. Navigazione tra i brani:

```
public void playNextInPlaylist() {
    if (currentPlaylist == null || currentPlaylist.getSongs().isEmpty()) {
        return;
    }

    int nextIndex = currentPlaylistIndex + 1;
    if (nextIndex >= currentPlaylist.getSongs().size()) {
        nextIndex = 0; // Loop back to start
    }

    playTrackAtIndex(nextIndex);
}
```

La navigazione tra i brani all'interno di una playlist è gestita dal metodo playNextInPlaylist(). Questo metodo verifica innanzitutto che esista una playlist valida e che contenga almeno un brano. Calcola quindi l'indice del prossimo brano da riprodurre; se si raggiunge la fine della playlist, l'indice viene riportato a zero, permettendo così una riproduzione ciclica. Infine, viene avviata la riproduzione del brano corrispondente al nuovo indice. Questo sistema consente all'utente di scorrere facilmente tra i brani della playlist, garantendo un'esperienza di ascolto continua e senza interruzioni.

4.1.4 Sistema di traduzioni

Il sistema di traduzioni è stato implementato per supportare il multilinguismo nell'applicazione, permettendo agli utenti di cambiare la lingua dell'interfaccia in modo dinamico. L'architettura segue un pattern a strati, separando la logica di business dall'accesso ai dati e dalla gestione dell'interfaccia utente.

1. **Gestione delle Traduzioni:**

Il TranslationsController implementa il pattern Singleton per garantire un unico punto di accesso alle traduzioni. All'inizializzazione, carica la lingua corrente dalle preferenze dell'utente e aggiorna il modello delle traduzioni.

```java
public class TranslationsController {
    private static TranslationsController instance;
    private final TranslationsBusinessInterface translationsModel;
    private final PreferencesBusinessInterface preferencesModel;
    private final StringProperty currentLanguage = new SimpleStringProperty();

    private TranslationsController() {
        preferencesModel = PreferencesModel.getInstance();
        translationsModel = TranslationsModel.getInstance();

        String currentLanguage = preferencesModel.getCurrentLanguage();
        translationsModel.changeLanguage(currentLanguage);
        this.currentLanguage.set(currentLanguage);
    }

    public static TranslationsController getInstance() {
        if (instance == null) {
            instance = new TranslationsController();
        }
        return instance;
    }
}
```

1. **Integrazione con l'Interfaccia Utente:**

L'applicazione principale gestisce il cambio di lingua attraverso un listener sulla proprietà currentLanguage. Quando la lingua viene cambiata, l'interfaccia utente viene ricaricata con le nuove traduzioni.

```java
public class AudioPlayerApp extends Application {
    private TranslationsController translations;

    @Override
    public void start(Stage primaryStage) {
        translations = TranslationsController.getInstance();
        loadUI();

        translations.currentLanguageProperty().addListener((obs, oldVal, newVal) -> {
            if (newVal != null && !newVal.equals(oldVal)) {
                try {
                    FXMLLoader loader = new FXMLLoader(getClass().getResource("/fxml/PlayerView.fxml"));
                    loader.setResources(ResourceBundle.getBundle("i18n.labels"));
                    loadUI();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        });
    }
}
```

Questo sistema di traduzioni offre diversi vantaggi:

- Separazione chiara delle responsabilità tra i vari componenti
- Facilità di aggiunta di nuove lingue
- Supporto per il cambio dinamico della lingua
- Persistenza della preferenza linguistica dell'utente
- Gestione efficiente delle risorse di traduzione

4.1.5 Pattern Singleton nell'Applicazione

Il pattern Singleton è stato implementato in diverse parti dell'applicazione per garantire che ci sia una sola istanza di classi critiche che gestiscono risorse condivise o funzionalità centrali. Ecco i principali casi d'uso:

Gestione del Player Audio (MP3Player)

La classe MP3Player implementa il Singleton per garantire che ci sia una sola istanza del player audio in esecuzione. Questo è fondamentale perché:

- Gestisce una singola risorsa audio
- Mantiene lo stato di riproduzione corrente
- Coordina le operazioni di play/pause/stop
- Gestisce il volume e altre impostazioni audio

Gestione delle Preferenze (PreferencesModel)

Il PreferencesModel utilizza il Singleton per gestire le preferenze dell'utente in modo centralizzato:

- Mantiene un'unica fonte di verità per le impostazioni
- Garantisce coerenza nell'accesso alle preferenze
- Gestisce il salvataggio e il caricamento delle preferenze

Gestione delle Traduzioni (TranslationsController)

Il TranslationsController implementa il Singleton per:

- Gestire centralmente le traduzioni dell'interfaccia
- Mantenere coerenza nella lingua selezionata
- Coordinare il cambio di lingua in tutta l'applicazione

Gestione delle Playlist (PlaylistManager)

Il PlaylistManager utilizza il Singleton per:

- Gestire centralmente le playlist dell'utente
- Coordinare le operazioni di creazione e modifica delle playlist
- Mantenere una lista coerente delle playlist disponibili

Caratteristiche Comuni dell'Implementazione

Tutte le implementazioni del Singleton nell'applicazione seguono un pattern comune:

1. Costruttore privato per impedire l'istanziazione diretta
2. Variabile statica instance per mantenere l'unica istanza
3. Metodo getInstance() per accedere all'istanza
4. Metodo resetInstance() per testing e reinizializzazione

Capitolo 5

5.1 Test e validazione

Per garantire l'affidabilità, la qualità e la stabilità dell'applicazione, sono stati condotti diversi test funzionali e non funzionali. Le verifiche hanno riguardato aspetti chiave come la riproduzione audio, la gestione delle playlist, il controllo del volume, la navigazione tra brani, la persistenza dei dati e l'interfaccia utente. Ogni test è stato strutturato con scenari precisi, criteri di passaggio e risultati attesi, al fine di validare il comportamento dell'applicazione nelle condizioni d'uso previste.

5.1.1 Test di Riproduzione Audio

Test di Caricamento File
ID Test: T001
Descrizione: Verifica del caricamento di un file audio
Prerequisiti: File MP3 valido presente nel sistema
Procedura:

1. Aprire l'applicazione
2. Cliccare su "Apri"
3. Selezionare un file MP3
4. Verificare che il file venga caricato correttamente

Risultato Atteso: Il file viene caricato e le informazioni (titolo, durata) vengono visualizzate
Criteri di Passaggio:

- Il file viene caricato senza errori
- Le informazioni del file vengono visualizzate correttamente
- Il player è pronto per la riproduzione

Test di Riproduzione Base
ID Test: T002
Descrizione: Verifica della riproduzione di un file audio
Prerequisiti: File MP3 caricato nel player
Procedura:

1. Caricare un file MP3
2. Cliccare sul pulsante Play
3. Verificare la riproduzione

Risultato Atteso: Il file inizia a riprodursi
Criteri di Passaggio:

- L'audio viene riprodotto correttamente
- Il pulsante Play cambia in Pause
- La barra di avanzamento si muove

Test di Pausa
ID Test: T003
Descrizione: Verifica della funzionalità di pausa
Prerequisiti: File MP3 in riproduzione
Procedura:

1. Avviare la riproduzione
2. Cliccare sul pulsante Pause
3. Attendere 5 secondi
4. Cliccare nuovamente Play

Risultato Atteso: La riproduzione si interrompe e riprende dallo stesso punto
Criteri di Passaggio:

- La riproduzione si interrompe
- Il pulsante cambia in Play
- La riproduzione riprende dallo stesso punto

5.1.2 Test di Gestione Playlist

Creazione Playlist
ID Test: T004
Descrizione: Verifica della creazione di una nuova playlist
Procedura:

1. Cliccare su "Nuova Playlist"
2. Inserire un nome per la playlist
3. Confermare la creazione

Risultato Atteso: La playlist viene creata e visualizzata nella lista
Criteri di Passaggio:

- La playlist viene creata
- Il nome viene salvato correttamente
- La playlist appare nella lista delle playlist

Aggiunta Canzoni alla Playlist
ID Test: T005
Descrizione: Verifica dell'aggiunta di canzoni a una playlist
Prerequisiti: Playlist esistente, file MP3 disponibili
Procedura:

1. Selezionare una playlist
2. Cliccare su "Aggiungi Canzone"
3. Selezionare un file MP3
4. Verificare l'aggiunta

Risultato Atteso: La canzone viene aggiunta alla playlist
Criteri di Passaggio:

- La canzone appare nella playlist
- Le informazioni della canzone sono corrette

5.1.3 Test di Controllo Volume

Regolazione Volume
ID Test: T006
Descrizione: Verifica della regolazione del volume
Prerequisiti: File in riproduzione
Procedura:

1. Avviare la riproduzione
2. Muovere lo slider del volume
3. Verificare il cambiamento del volume

Risultato Atteso: Il volume cambia in base alla posizione dello slider
Criteri di Passaggio:

- Il volume cambia gradualmente
- Il valore del volume viene aggiornato

Test Mute
ID Test: T007
Descrizione: Verifica della funzionalità mute
Prerequisiti: File in riproduzione
Procedura:

1. Avviare la riproduzione
2. Cliccare sul pulsante Mute
3. Verificare l'assenza di audio
4. Cliccare nuovamente per ripristinare

Risultato Atteso: L'audio viene disattivato e riattivato
Criteri di Passaggio:

- L'audio viene disattivato
- L'icona mute cambia
- L'audio viene ripristinato al livello precedente

5.1.4 Test di Navigazione

Cambio Traccia
ID Test: T008
Descrizione: Verifica del cambio traccia
Prerequisiti: Playlist con più canzoni

Procedura:
1. Avviare la riproduzione
2. Cliccare su "Prossima Traccia"
3. Verificare il cambio

Risultato Atteso: La riproduzione passa alla traccia successiva
Criteri di Passaggio:
- La nuova traccia inizia a riprodursi
- Le informazioni della traccia vengono aggiornate
- La barra di avanzamento si resetta

Riproduzione Casuale
ID Test: T009
Descrizione: Verifica della riproduzione casuale
Prerequisiti: Playlist con più canzoni
Procedura:
1. Attivare la modalità casuale
2. Avviare la riproduzione
3. Verificare l'ordine delle tracce

Risultato Atteso: Le tracce vengono riprodotte in ordine casuale
Criteri di Passaggio:
- L'ordine delle tracce è casuale
- Non ci sono ripetizioni immediate
- La riproduzione continua senza errori

5.1.5 Test di Persistenza Dati

Salvataggio Playlist
ID Test: T010
Descrizione: Verifica del salvataggio delle playlist
Prerequisiti: Playlist modificata
Procedura:
1. Modificare una playlist
2. Chiudere l'applicazione
3. Riaprire l'applicazione
4. Verificare il caricamento delle playlist

Risultato Atteso: Le playlist vengono salvate e ricaricate correttamente
Criteri di Passaggio:
- Le playlist vengono salvate
- I dati vengono ricaricati correttamente
- Le modifiche sono persistenti

5.1.6 Test di Interfaccia Utente

Responsività UI
ID Test: T011
Descrizione: Verifica della responsività dell'interfaccia
Procedura:

1. Ridimensionare la finestra
2. Verificare il comportamento dei componenti

Risultato Atteso: L'interfaccia si adatta al ridimensionamento
Criteri di Passaggio:

- I componenti si ridimensionano correttamente
- Non ci sono sovrapposizioni
- L'interfaccia rimane utilizzabile

5.1.7 Test di Performance

Caricamento File Grandi
ID Test: T012
Descrizione: Verifica del caricamento di file di grandi dimensioni
Prerequisiti: File MP3 di grandi dimensioni
Procedura:

1. Caricare un file MP3 di grandi dimensioni
2. Verificare il tempo di caricamento
3. Verificare la riproduzione

Risultato Atteso: Il file viene caricato e riprodotto senza problemi
Criteri di Passaggio:

- Il caricamento avviene in tempi ragionevoli
- La riproduzione è fluida
- Non ci sono problemi di memoria

5.2 Risultati dei Test

ID Test	Descrizione	Risultato	Data Test	Note
T001	Test di Caricamento File	PASSATO	2025-06-05	Caricamento file MP3 completato con successo
T002	Test di Riproduzione Base	PASSATO	2025-06-05	Riproduzione audio funzionante
T003	Test di Pausa	PASSATO	2025-06-05	Funzionalità pausa/riprendi operativa
T004	Creazione Playlist	PASSATO	2025-06-05	Creazione playlist completata
T005	Aggiunta Canzoni alla Playlist	PASSATO	2025-06-05	Aggiunta canzoni funzionante
T006	Regolazione Volume	PASSATO	2025-06-05	Controllo volume operativo
T007	Test Mute	PASSATO	2025-06-05	Funzionalità mute operativa
T008	Cambio Traccia	PASSATO	2025-06-05	Navigazione tracce funzionante
T009	Riproduzione Casuale	PASSATO	2025-06-05	Modalità shuffle operativa
T010	Salvataggio Playlist	PASSATO	2025-06-05	Persistenza dati verificata
T011	Responsività UI	PASSATO	2025-06-05	Interfaccia responsive
T012	Caricamento File Grandi	PASSATO	2025-06-05	Performance ottimale

Riepilogo:
- Totale Test: 14
- Test Passati: 14
- Test Falliti: 0
- Percentuale Successo: 100%

Capitolo 6

6.1 Conclusione

Realizzare questo progetto di Music Player è stata per noi una sfida stimolante e, allo stesso tempo, un'occasione preziosa di crescita personale e professionale. Lavorare in coppia ci ha permesso di confrontarci costantemente, di unire le nostre competenze e di imparare a collaborare in modo efficace, affrontando insieme sia le difficoltà tecniche che quelle organizzative.

Durante lo sviluppo abbiamo approfondito tecnologie come JavaFX, ma soprattutto abbiamo imparato quanto sia importante progettare un'architettura chiara e modulare, che renda il codice facilmente estendibile e manutenibile. Ogni funzionalità, dalla gestione delle playlist al controllo del volume, è stata pensata per offrire un'esperienza utente fluida e intuitiva, ma anche per essere un esercizio concreto di buona programmazione.

Siamo particolarmente orgogliosi di aver realizzato un'applicazione che non solo funziona, ma che rispecchia anche la nostra passione per la musica e per il software ben fatto. Sappiamo che ci sono ancora molti margini di miglioramento e tante funzionalità che si potrebbero aggiungere, ma crediamo che questo progetto rappresenti una solida base su cui costruire in futuro.

In conclusione, questa esperienza ci ha insegnato che il lavoro di squadra, la curiosità e la voglia di mettersi in gioco sono ingredienti fondamentali per affrontare con successo qualsiasi sfida, dentro e fuori dall'università.