Anno accademico 2022/2023
CORSO DI PROGETTAZIONE E ALGORITMI

WePlay

Documentazione di progetto

Irene Bravo matr. 1071820
Samuel Locatelli matr. 1054674
Greta Marrone matr. 1058513


WM

Indice

Elenco delle figure

Capitolo 1

Iterazione 0

1.1 Requisiti

Si vuole realizzare un sistema software che offra un servizio di sharing musicale, integrato con le funzionalità tipiche di un social network. Durante la fase di registrazione, l'utente sarà tenuto ad inserire una lista di generi musicali preferiti. L'algoritmo proporrà quindi all'utente i brani più popolari basati sulle sue preferenze e gli consetirà di esplorare le sue affinità musicali. Gli attori coinvolti sono i seguenti:

- **Utenti:** coloro che desiderano ascoltare musica, conoscere e condividere le proprie attività musicali con gli amici;
- **Artisti:** sia indipendenti che non, per pubblicare e gestire la propria musica.

Per poter usufruire dei servizi forniti dal sistema l'utente deve registrarsi e creare un proprio profilo personale. Una volta completata la registrazione, verrà indirizzato alla homepage del sistema dove potrà visualizzare i brani più popolari appartenenti ai generi musicali di suo interesse, avrà la possibilità di effettuare ricerche specifiche di brani e accedere alla sezione delle impostazioni personali e social. Nel momento in cui viene selezionato un brano per il download, saranno fornite tutte le informazioni relative ad esso, tra cui titolo, artista, album, anno di pubblicazione, genere di appartenenza, numero di ascolti, durata del brano e casa discografica

Il software offre anche la possibilità di mettere "like" ai brani e di creare playlist personalizzate. Nella propria libreria musicale saranno quindi presenti le playlist dell'utente e la playlist dei brani preferiti. Inoltre l'interfaccia social permette di aggiungere gli amici e di chattare con essi.

La funzionalità principale offerta dal servizio è l'algoritmo "Discover", che permette all'utente di trovare i brani più apprezzati dalle persone che ascoltano gli stessi generi musicali, al fine di permettergli di scoprire sempre nuove canzoni. Inoltre, l'algoritmo è anche in grado di proporre

L'artista, a differenza dell'utente ascoltatore, può caricare brani singoli e album inserendo i dati relativi alla propria produzione artistica.

L'utente, per poter usufruire dei servizi forniti, deve registrarsi e creare un proprio profilo personale. Una volta iscritto, verrà indirizzato alla homepage che mostra i brani più popolari dei generi preferiti dall'utente, la possibilità di cercare un brano specifico e la sezione di impostazioni personali e social. Quando viene selezionato il brano da scaricare, vengono fornite tutte le informazioni relative (titolo, artista, album, anno di uscita, genere di appartenenza, numero di ascolti, durata del brano, casa discografica).

Il software offre anche la possibilità di mettere "like" ai brani e di creare playlist personalizzate. Nella propria libreria saranno quindi presenti le playlist dell'utente e la playlist dei brani preferiti. L'interfaccia social permette invece di aggiungere gli amici e di chattare con essi.

La funzionalità principale offerta dal servizio è l'algoritmo "Discover", che consente di trovare i brani più apprezzati dalle persone che ascoltano gli stessi generi musicali dell'utente così da permettergli di scoprire sempre nuove canzoni.

L'artista, a differenza dell'utente ascoltatore, può caricare brani singoli e album inserendo i dati relativi alla propria produzione artistica.

1.2 Studio di fattibilità

Procedendo con una valutazione dei costi e dei benefici della possibile realizzazione del presente sistema, è possibile concludere che i costi principali sono relativi alla funzionalità di Discover e brani consigliati.

Per realizzare questa funzionalità è necessaria l'implementazione di un algoritmo che vada a studiare i gusti e le preferenze di ogni singolo utente e, dopo la preliminare fase di raccolta dati, proporrà all'utente una serie di brani in linea con le sue preferenze.

Per quanto riguarda invece la fattibilità in termini di tecnologie e strumenti per la realizzazione del sistema il costo principale da considerare è quello relativo allo sviluppo del software e il servizio di hosting sia della piattaforma che del database.

1.3 Casi d'uso

Di seguito vengono riportati i casi d'uso raggruppati in base all'attore coinvolto. Come anticipato gli attori in gioco sono i seguenti:

- Utente ascoltatore;
- Artista;

1.3.1 Casi d'uso utente

- **UC1: Sign up**

  L'utente può registrarsi inserendo le sue informazioni personali.

- **UC2: Sign in**

  L'utente può accedere al proprio profilo personale inserendo nome utente e password.

- **UC3: Sign out**

  L'utente può disconnettere il proprio profilo personale dal sistema.

- **UC4: Cerca brano**

  L'utente può digitare nella barra di ricerca per cercare un brano dal titolo.

- **UC5: Cerca album**

  L'utente può digitare nella barra di ricerca per cercare un album dal titolo.

- **UC6: Cerca Artista**

  L'utente può digitare nella barra di ricerca per cercare un artista dal nome.

- **UC7: Scarica brano**

  L'utente può scaricare il brano.

- **UC8: Like al brano**

  L'utente può esprimere il proprio interesse per un brano mettendo like.

- **UC9: Aggiungi brano a playlist**

  L'utente può aggiungere alla lista di brani della playlist un brano selezionato.

- **UC10: Cerca Utente**

  L'utente può cercare un utente per nome dalla barra di ricerca apposita (cerca album/artista).

- **UC11: Aggiungi Utente**

  L'utente può aggiungere alla propria lista amici un utente selezionato.

- **UC12: Visualizza informazioni profilo**

  L'utente può consultare le proprie informazioni personali inserite in fase di registrazione.

- **UC13: Modifica profilo**

  L'utente può modificare le proprie informazioni personali inserite in fase di registrazione.

- **UC14: Elimina profilo**

  L'utente può eliminare il proprio profilo dal sistema.

- **UC15: Visualizza playlist**

  L'utente può consultare tutte le proprie playlist.

- **UC16: Crea nuova playlist**

  L'utente può creare una nuova playlist nella quale aggiungere brani.

- **UC17: Elimina playlist**

  L'utente può eliminare una delle proprie playlist.

- **UC18: Modifica playlist**

  L'utente può modificare una delle proprie playlist, per esempio modificando il nome o rimuovendo brani.

- **UC19: "Discover"** L'utente può consultare brani scelti per lui dal sistema e amici suggeriti in base alle sue preferenze musicali.

1.3.2 Casi d'uso artista

- **UC20: Crea pagina artista**

  L'artista per poter pubblicare musica deve creare una pagina artista inserendo le proprie informazioni personali.

- **UC21: Visualizza pagina artista**

  L'artista può consultare la propria pagina artista, dove vengono visualizzati i propri brani/album pubblicati.

- **UC22: Aggiungi brano**

  L'artista può pubblicare un nuovo brano inserendo le informazioni relative.

- **UC23: Aggiungi album**

  L'artista può pubblicare un nuovo album contenente i brani scelti.

- **UC24: Personalizza pagina artista**

  L'artista può decidere quali brani e album mettere in evidenza nella propria pagina artista.

- **UC25: Consulta anagrafica**

  L'artista può consultare l'anagrafica relativa agli ascolti e ai like dei brani e album pubblicati.

1.3.3 Dettaglio casi d'uso

Di seguito verranno analizzati più nel dettaglio i casi d'uso sopra citati, identificandone descrizione, attore e passi principali.

Gli utenti avranno la possibilità di registrarsi al sito per poter fruire dei servizi di streaming di musica. Al momento della registrazione sarà necessario inserire alcune informazioni personali di base come nome, cognome, email. Inoltre si dovrà inserire un nome utente identificativo e una password, necessari per il login. Nel caso in cui l'utente sia già registrato sarà necessario inserire solo questi campi per accedere al sistema.

Al termine dell'iscrizione è possibile navigare nella Home page, la quale si suddivide in varie sezioni.

È possibile cercare un brano dalla barra di ricerca (tramite il titolo, nome dell'artista o nome dell'album) e scaricarlo.

Una volta scelto il brano, per ognuno di essi vengono visualizzate alcune informazioni come il titolo del brano, il nome dell'artista e l'etichetta discografica di riferimento.

È offerta la possibilità di mettere like al brano e creare playlist personalizzate con i propri brani preferiti: visualizzando la propria libreria, infatti, è possibile visualizzare i Brani Preferiti e le proprie playlist.

Come anticipato il sistema offre anche funzionalità di un social network: infatti dalla barra di ricerca è possibile cercare per nome utente i propri amici e aggiungerli, in modo tale da poter condividere le proprie preferenze musicali.

Una delle funzionalità principali offerte dal sistema è chiamata Discover e consiste nel poter usufruire di playlist create ad hoc dal sistema per l'utente, tramite lo studio delle sue preferenze.

Gli artisti potranno gestire e pubblicare la propria musica tramite varie funzionalità offerte dal sistema. Al momento della registrazione bisogna seguire la procedura classica di sign-in, in seguito in una sezione apposita è offerta la possibilità di creare la pagina artista, all'interno della quale verrà inserito il nome che apparirà pubblicamente, insieme ad altre informazioni personali.

Dopo aver creato il proprio profilo artista si potrà accedere alla propria pagina per pubblicare le proprie tracce, album, playlist, personalizzandola. Ogni volta che viene pubblicato un brano è possibile inserire eventuali nomi di collaboratori, artisti che hanno partecipato al brano e scrittori di testo e musica. Ogni artista può scegliere la data dalla quale rendere accessibile il brano o l'album.

1.3.4 Diagramma dei casi d'uso

In figura 1.1 è stato riportato il diagramma dei casi d'uso, dove sono stati inseriti quelli già precedentemente descritti mettendoli in correlazione con il corrispettivo attore.

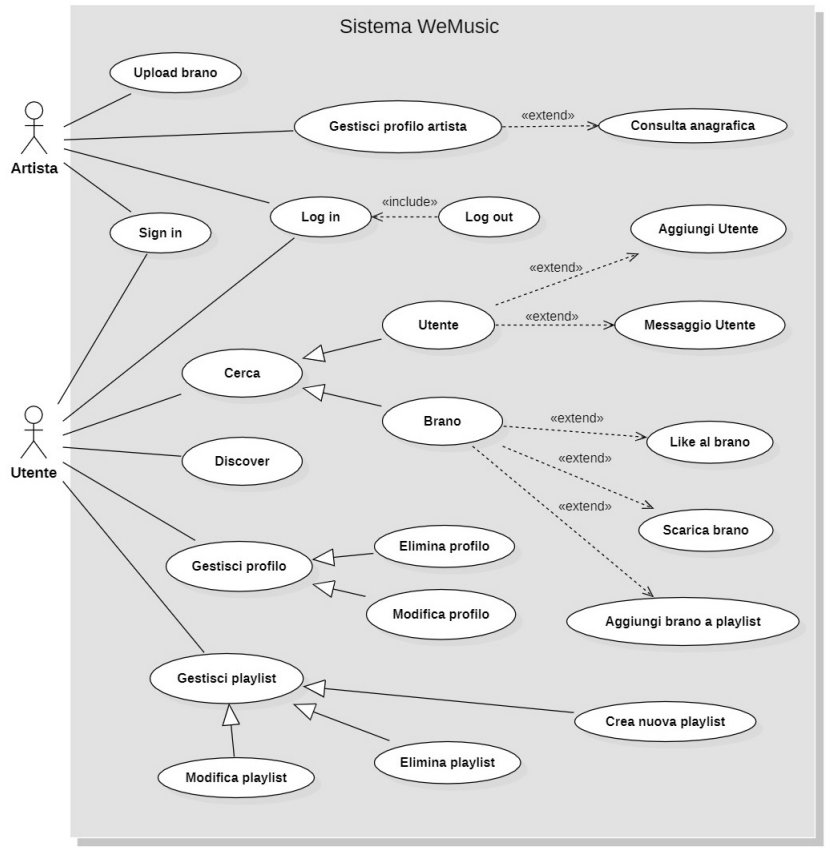

Figura 1.1: UML Use Cases Diagram

1.4 Architettura del sistema

L'architettura del sistema è stata formalizzata attraverso la rappresentazione tramite Deployment Diagram, diagramma di tipo statico sviluppato al fine di modellizzare e descrivere un sistema in termini di risorse hardware e di relazioni fra di esse. Vengono sviluppatidue Deployment Diagram differenti:

Il primo, riportato in figura 1.2, rappresenta il sistema attraverso una notazione a stile libero, mentre il secondo, riportato in figura 1.3, rappresenta la topologia del sistema tramite lo stile UML.

1.4.1 Deployment diagram – Informal

In figura 1.2 viene evidenziata la componente principale del sistema, il sistema gestore (WeMusic) il Web Server (sviluppato tramite il framework Django) che, tramite una connessione ad internet, permette agli utenti di poter consultare il sito.

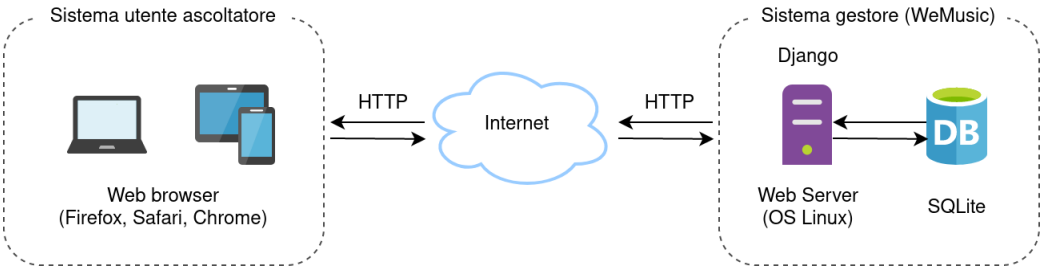

Figura 1.2: Architettura

1.4.2 Deployment diagram – UML

In figura 1.3 vengono evidenziati due nodi principali: un lato client che gestirà il front-end del sistema WeMusic, e un lato server relativo al back-end e alla gestione del DataBase, in modo da velocizzare e semplificare l'interazione tra logica e dati.

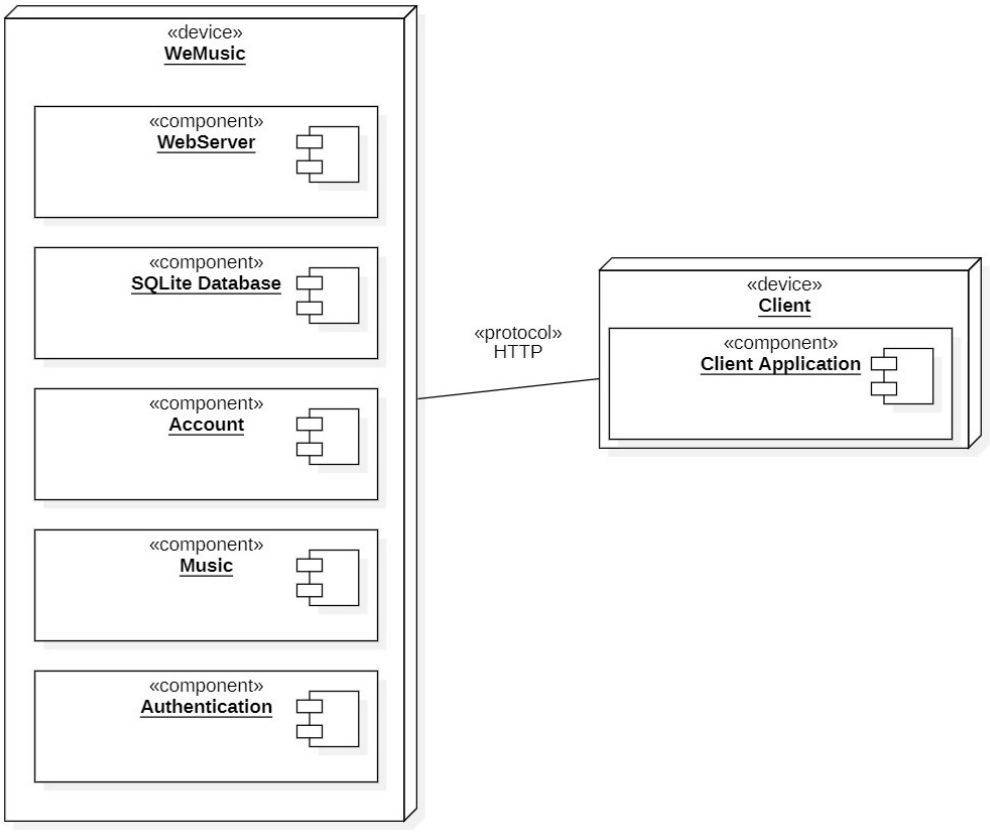

Figura 1.3: Deployment Diagram

Capitolo 2

Iterazione 1

2.1 Introduzione

L'iterazione 1 ha come scopo quello di identificare le componenti dal modello dei casi d'uso, applicando le euristiche di early design: verrà perfezionata la specifica dei componenti progettati durante l'iterazione 0 al fine di da definire meglio l'architettura software. È stato costruito lo scheletro dell'applicazione tramite la specifica delle differenti classi popolate nelle seguenti iterazioni.

Inizialmente, durante lo sviluppo dell'iterazione 0, il sistema nella sua totalità è stato visualizzato come una unica componente e sono stati introdotti gli attori, ognuno dei quali definito come un'entità esterna. In seguito sono stati sviluppati tutti i casi d'uso che riassumono il funzionamento del sistema, i quali verranno, nell'Iterazione 1, raccolti e raggruppati secondo un preciso criterio e affinità.

Vengono inoltre introdotti componenti e sottocomponenti di controllo di ciascun gruppo, o alternativamente componenti di dati; per ognuna di esse nascono parallelamente le classi candidate e le relazioni che le legano.

Per ogni caso d'uso che ha una interazione diretta con un attore esterno viene introdotta un'interfaccia per le operazioni visibili esternamente, ovvero le API, offerte o richieste dal componente o sottocomponente corrispondente, in base alla direzione dell'interazione.

Ogni variabile input da un attore, specificata nella descrizione di un caso d'uso, definisce un parametro di input dell'operazione dell'interfaccia corrispondente. Ogni variabile output per un attore specificata nella descrizione di un caso d'uso definisce un parametro di ritorno dell'operazione dell'interfaccia corrispondente (se modalità sincrona) o un input di un'operazione dell'interfaccia di callback (se modalità asincrona) dell'attore.

Viene scomposto il sistema in sottosistemi e componenti, applicando pattern e stili architetturali, per poi distribuire le componenti su nodi computazionali basati su uno sviluppo fisico del sistema.

E' necessario specificare che l'architettura di Django si basa sul design pattern MVC, acronimo di Model View Controller. Lo scopo di questa scelta di design è la suddivisione della logica in tre parti fondamentali favorendo modularità, sviluppo collaborativo e riutilizzo del codice oltre a rendere l'applicazione più flessibile e scalabile.

Django interpreta il pattern MVC a modo suo, in modo tale che il Controller risulti in realtà il framework in sé, le View non scelgano come i dati vadano mostrati bensì quali dati mostrare e il come mostrarli viene in realtà deciso nei Template.

Siccome la parte del Controller è gestita dal framework e la View è rappresentata dai Template, Django utilizza un framework MTV, ovvero Model Template View.

- **M - Model:** "data access layer" - Definisce la struttura delle entità (le classi) del nostro sito, tradotte poi in tabelle nel database.
- **T - Template:** "presentation layer" - Descrive come i dati vadano mostrati nelle pagine web.
- **V - View:** "business logic layer" - Gestisce le richieste e le risposte HTTP, dispone della logica per sapere a quali dati accedere tramite i model e delega la formattazione della risposa ai template, contenendo quindi i metodi.

2.2 Casi d'uso

I casi d'uso vengono raggruppati in tre macro categorie in base a un criterio di affinità: i primi sono relativi all'autenticazione dell'utente, i secondi sono relativi a tutto ciò che riguarda l'ambito musicale, ed infine quelli relativi all'account personale e lato social. Di seguito un elenco dettagliato dei casi d'uso suddivisi come anticipato.

2.2.1 Autenticazione

- **UC1:** Sign up
- **UC2:** Sign in
- **UC3:** Sign out

2.2.2 Musica

- **Brano:** in questa sezione rientrano tutti i casi d'uso relativi ai brani.
  - **UC4:** Cerca brano
  - **UC5:** Cerca album
  - **UC6:** Cerca Artista
  - **UC7:** Scarica brano
  - **UC8:** Like al brano
  - **UC19:** "Discover"
- **Playlist:** in questa sezione rientrano tutti i casi d'uso relativi alla creazione e gestione delle playlist.
  - **UC9:** Aggiungi brano a playlist
  - **UC15:** Visualizza playlist
  - **UC16:** Crea nuova playlist
  - **UC17:** Elimina playlist
  - **UC18:** Modifica playlist
- **Artista:** in questa sezione rientrano tutti i casi d'uso relativi alla gestione del profilo artista.
  - **UC20:** Crea pagina artista

- **UC21:** Visualizza pagina artista
- **UC22:** Aggiungi brano
- **UC23:** Aggiungi album
- **UC24:** Personalizza pagina artista
- **UC25:** Consulta anagrafica

2.2.3 Account

- **Personale:** in questa sezione rientrano tutti i casi d'uso relativi alla gestione delle informazioni nel proprio profilo personale.
  - **UC12:** Visualizza informazioni profilo
  - **UC13:** Modifica profilo
  - **UC14:** Elimina profilo
- **Amici:** in questa sezione rientrano tutti i casi d'uso relativi alla parte social.
  - **UC10:** Cerca Utente
  - **UC11:** Aggiungi Utente

2.3 UML Data Class Diagram

Il Class Diagram in UML permette di descrivere il sistema visualizzando i diversi tipi di oggetti all'interno di esso e le relazioni statiche che esistono fra loro: sono descritte in maniera più approfondita le tre componenti già precedentemente analizzate, ovvero Account, Musica, Autenticazione e Discover. In questo caso viene rappresentato in unico diagramma il Data Class Diagram, il quale descrive le classi che compongono la struttura dell'applicazione e le le entità presenti al suo interno, e il Package/IF Diagram, quindi sono stati inserite le funzioni implementate. nel sistema.

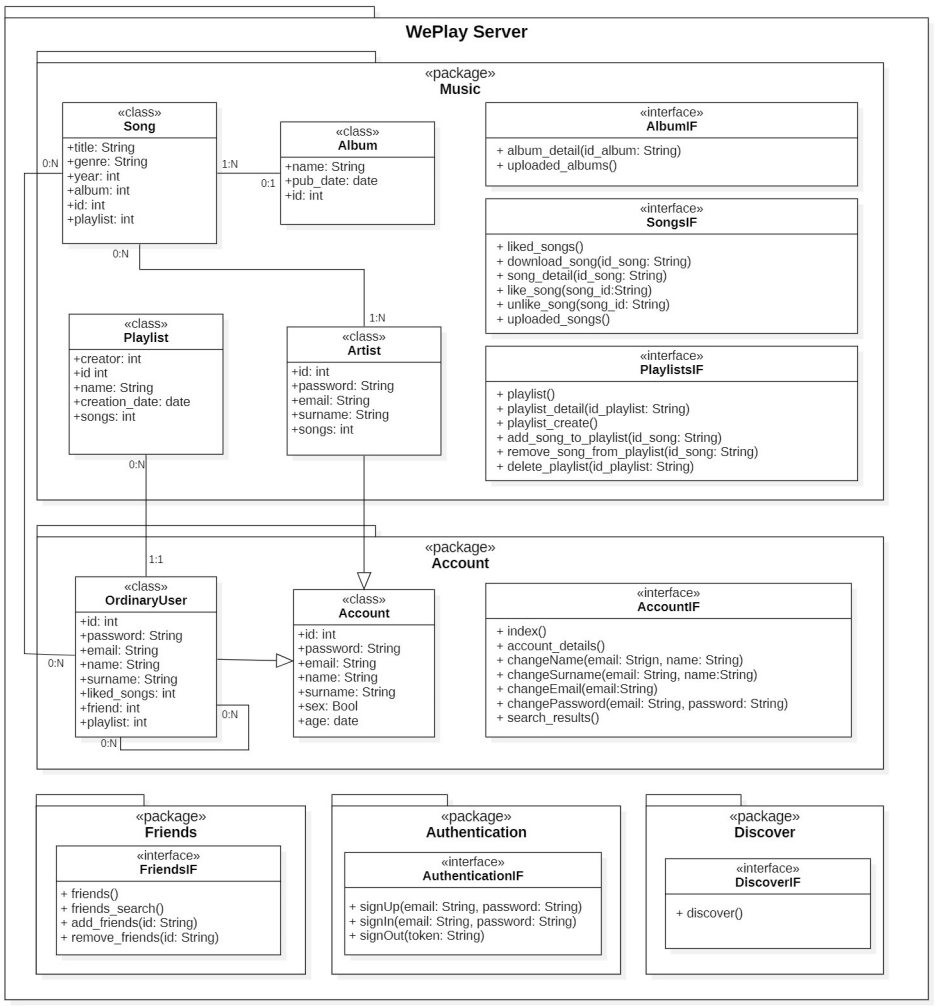

Figura 2.1: UML Class Diagram

2.4 UML Component Diagram

Il Component Diagram in UML ha come obiettivo quello di mostrare la struttura del sistema software, descrivendo i singoli componenti, le relative interfacce e le dipendenze.

2.4.1 Black Box - Top level

La rappresentazione black box del sistema è stata gestita partendo da una una componente centrale, il server del sistema, alla quale si collegano il database e la parte front-end. Le componenti principali sono: Account, Authentication, Music (che include Brani, Playlist, Album), Discover e Friends.

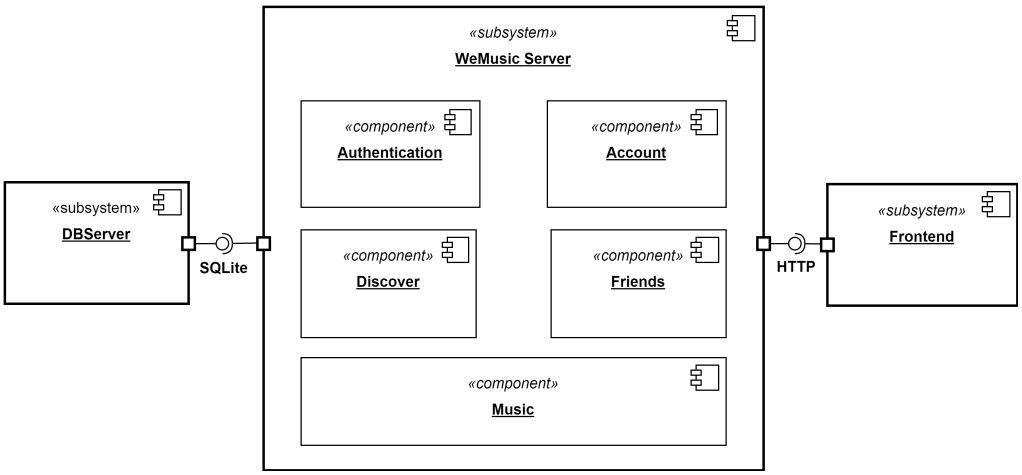

Figura 2.2: UML Component Diagram

2.4.2 White Box - Top level

La rappresentazione white box del sistema rappresenta anche le componenti interne e le interfacce tramite le quali comunicano tra loro.

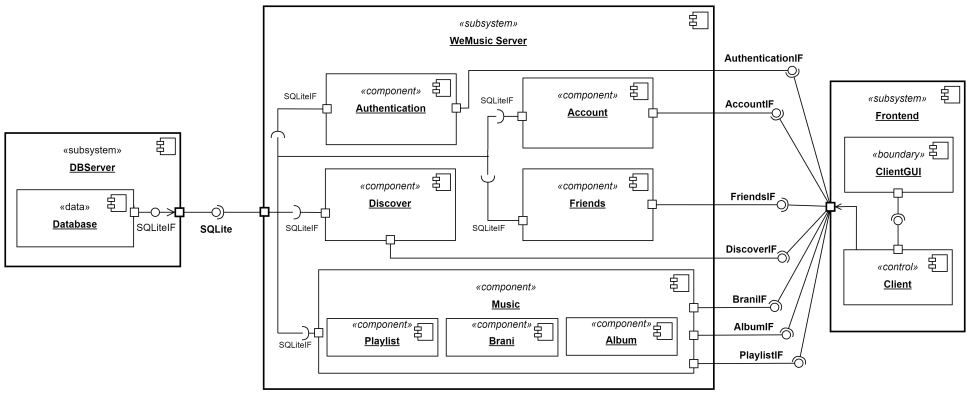

Figura 2.3: UML Component Diagram

Capitolo 3

Iterazione 2

3.1 Introduzione

Nella secconda iterazione sono stati implementati la maggior parte dei casi d'uso: viene proposta un'analisi dettagliata al fine di specificare le seguenti informazioni:

- **Attori:** vengono specificati quali attori sono coinvolti nel caso d'uso in analisi.
- **Precondizioni:** condizioni che devono necessariamente essere verificate per il corretto funzionamento del sistema nel caso d'uso in analisi.
- **Postcondizioni:** condizioni che saranno verificate a seguito di un corretto funzionamento del sistema nel caso d'uso in analisi.
- **Sequenza base:** riassume gli step base del caso d'uso in analisi.
- **Sequenza in caso di eccezioni:** specifica i casi d'eccezione possibili e a che passo della Base Sequence si viene riportati.

Dopo aver analizzato nel dettaglio i casi d'uso si procede con la parte di Testing, quindi analisi statica e dinamica tramite Electron. (...)

3.2 Implementazione

Di seguito sono elencati i casi d'uso implementati nel sistema WeMusic. Si offre, per ognuno, una breve descrizione dello stesso al fine di riassumere la sua funzionalità; inoltre è evidenziato il flow di esecuzione del sito e le varie alterntive che si propongono nel modello. Infine, vengono specificate eventuali informazioni relative a precondizioni, post condizioni e trigger dello stesso, e le possibili eccezioni nel caso avvenga una esecuzione scorretta. Seguendo la modellazione proposta in seguito, i casi d'uso relativi all'Autenticazione sono stati completamente implementati, come anche la maggior parte di quelli facenti parte del gruppo Musica e Amici.

Bootstrap Per il frontend è stato utilizzato Bootstrap, un framework per frontend e web development semplice e veloce. Bootstrap include templates design basati su HTML e CSS per tipografia, forme, pulsanti, tabelle, navigazione, caroselli di immagini e molto altro, come anche plugin opzionali di JavaScript; è' molto semplice da usare e possiede un'alta compatibilità con i browser maggiormente usati.

3.2.1 UC1 – Sign up

Il caso d'uso di Sign up consente all'utente di registrare al sito e di poter usufruire dei servizi offerti dallo stesso. Il presente caso d'uso è attivo solo dal lato amministratore e non è ancora disponibile per gli utenti regolari, ma il suo funzionamento è stato comunque riepilogato di seguito.

- **Actor:** Utente base, Artista.
- **Precondition:** Nessuna.
- **Postcondition:** L'utente è registrato al sito, le sue informazioni vengono salvate nel database.
- **Base Sequence:**
  - **1.** L'utente inserisce nome utente e password.
  - **2.** L'utente ha completato la registrazione.
- **Exception Sequence:** Il nome utente scelto non è valido o è già in uso, o connessione fallita.
  - Nel caso di nome utente non valido o già in uso si ritorna al passo 1 della Base Sequence del presente caso d'uso.

- Nel caso di connessione fallita si ritorna al passo 1 della Base Sequence del presente caso d'uso.

3.2.2 UC2 – Sign in

Il caso d'uso di Sign in permette ad un utente già registrato nel sistema di accedere e di poter usufruire dei servizi offerti dallo stesso.

- **Actor:** Utente base, Artista.
- **Precondition:** L'utente deve essere già registrato.
- **Postcondition:** L'utente ha effettuato il log in perciò puo navigare la Home page del sito.
- **Base Sequence:**
  - **1.** L'utente inserisce nome utente e password.
  - **2.** L'utente clicca sul pulsante di log in.
  - **3.** L'utente ha effettuato correttamente l'accesso.
- **Exception Sequence:** La password inserita è errata o connessione fallita.
  - Nel caso di password errata si ritorna al passo 1 della Base Sequence del presente caso d'uso.
  - Nel caso di connessione fallita si ritorna al passo 1 della Base Sequence del caso d'uso **UC1**.

3.2.3 UC3 – Sign out

Il caso d'uso di Sign out permette all'utente di disconnettere il proprio profilo personale dal sistema.

- **Actor:** Utente base, Artista.
- **Precondition:** L'utente deve essere registrato e deve aver effettuato il log in.

- **Postcondition:** L'utente ha effettuato il log out, perciò non ha più accesso al sistema fino al log in.
- **Base Sequence:**
  - **1.** L'utente clicca il pulsante di Log out nell'interfaccia.
  - **2.** L'utente è disconnesso.
- **Exception Sequence:** Connessione fallita.
  - Nel caso di connessione fallita si ritorna al passo 1 della Base Sequence del caso d'uso **UC1**.

3.2.4 UC4 – Cerca brano

Il caso d'uso di Cerca brano permette di cercare un brano specifico dalla barra di ricerca tramite precise parole chiave, al fine di scaricarlo per poi riprodurlo in locale.

- **Actor:** Utente base.
- **Precondition:** L'utente deve aver effettuato il log in.
- **Postcondition:** L'utente ha cercato il brano, può dunque mettere like, aggiungerlo ad una playlist personale o scaricarlo. L'utente può scaricare il brano **(UC7)**, mettere like al brano **(UC8)**, aggiungerlo ad una playlist **(UC9)**.
- **Base Sequence:**
  - **1.** L'utente clicca sulla barra di ricerca apposita nell'interfaccia.
  - **2.** L'utente digita delle parole chiave per cercare il brano: titolo del brano, album, artista.
  - **3.** L'utente ha correttamente eseguito la ricerca: può navigare fra i risultati e selezionare il brano desiderato.
- **Exception Sequence:** Non esiste un brano che corrisponde ai criteri di ricerca o connessione fallita.
  - Nel caso di ricerca fallita si ritorna al caso 2 della Base Sequence del presente caso d'uso.

- Nel caso di connessione fallita si ritorna al passo 1 della Base Sequence del caso d'uso **UC1**.

3.2.5 UC5 – Cerca album

Il caso d'uso di Cerca album consiste nel cercare un album specifico dalla barra di ricerca tramite precise parole chiave, al fine di selezionare un brano al suo interno e scaricarlo per poi riprodurlo in locale.

- **Actor:** Utente base.
- **Precondition:** L'utente deve aver effettuato il log in.
- **Postcondition:** L'utente ha cercato l'album, può dunque navigare fra i brani che include lo stesso. L'utente può navigare fra i brani dell'album e scaricarli **(UC7)**, mettere like **(UC8)**, aggiungerli ad una playlist **(UC9)**.
- **Base Sequence:**
  - **1.** L'utente clicca sulla barra di ricerca apposita nell'interfaccia.
  - **2.** L'utente digita delle parole chiave per cercare l'album: titolo dell'album, artista.
  - **3.** L'utente ha correttamente eseguito la ricerca: può navigare fra i risultati e selezionare l'album desiderato.
- **Exception Sequence:** Non esiste un album che corrisponde ai criteri di ricerca o connessione fallita.
  - Nel caso di ricerca fallita si ritorna al caso 2 della Base Sequence del presente caso d'uso.
  - Nel caso di connessione fallita si ritorna al passo 1 della Base Sequence del caso d'uso **UC1**.

3.2.6 UC6 – Cerca artista

Il presente caso d'uso consiste nel cercare un artista specifico dalla barra di ricerca tramite precise parole chiave, al fine di selezionare un brano da lui pubblicato e scaricarlo per poi riprodurlo in locale.

- **Actor:** Utente base.
- **Precondition:** L'utente deve aver effettuato il log in.
- **Postcondition:** L'utente ha cercato l'artista, può dunque navigare nella pagina artista e scaricare i brani di interesse. L'utente può navigare fra i brani dell'artista e scaricarli **(UC7)**, mettere like **(UC8)**, aggiungerli ad una playlist **(UC9)**.
- **Base Sequence:**
  - **1.** L'utente clicca sulla barra di ricerca apposita nell'interfaccia.
  - **2.** L'utente digita delle parole chiave per cercare l'artista, ovvero il nome dell'artista. artista.
  - **3.** L'utente ha correttamente eseguito la ricerca: può navigare fra i risultati e selezionare l'artista desiderato.
- **Exception Sequence:** Non esiste un artista che corrisponde ai criteri di ricerca o connessione fallita.
  - Nel caso di ricerca fallita si ritorna al caso 2 della Base Sequence del presente caso d'uso.
  - Nel caso di connessione fallita si ritorna al passo 1 della Base Sequence del caso d'uso **UC1**.

3.2.7 UC7 – Scarica brano

Il presente caso d'uso consiste nello scaricare un brano al fine di poterlo riprodure in locale.

- **Actor:** Utente base.
- **Precondition:** L'utente deve aver effettuato il log in e individuato il brano desiderato.

- **Postcondition:** L'utente ha scaricato il brano scelto, quindi può ascoltarlo offline.
- **Base Sequence:**
  - **1.** L'utente seleziona il brano desiderato (da una playlist, dalla ricerca, ecc).
  - **2.** L'utente clicca sul pulsante di download.
  - **3.** L'utente attende il completamento del download.
  - **4.** L'utente ha correttamente scaricato il brano.
- **Exception Sequence:** Connessione fallita.
  - Nel caso di connessione fallita si ritorna al passo 1 della Base Sequence del caso d'uso **UC1**.

3.2.8 UC8 – Like al brano

Il presente caso d'uso consiste nel mettere like ad un brano; in questo modo è visibile nell'elenco dei propri brani preferiti.

- **Actor:** Utente base.
- **Precondition:** L'utente deve aver effettuato il log in e individuato il brano desiderato.
- **Postcondition:** L'utente ha aggiunto il brano ai suoi preferiti, quindi può visualizzarlo nell'elenco dei propri preferiti.
- **Base Sequence:**
  - **1.** L'utente seleziona il brano desiderato (da una playlist, dalla ricerca, ecc).
  - **2.** L'utente clicca sul pulsante di like.
  - **3.** L'utente ha completato correttamente l'operazione: il brano sarà visibile fra i brani preferiti.
- **Exception Sequence:** Connessione fallita.
  - Nel caso di connessione fallita si ritorna al passo 1 della Base Sequence del caso d'uso **UC1**.

3.2.9 UC9 – Aggiungi brano a playlist

Il presente caso d'uso consiste nell'aggiungere un brano ad una playlist già esistente.

- **Actor:** Utente base.
- **Precondition:** L'utente deve aver effettuato il log in e deve aver creato almeno una playlist, oltre che individuato il brano desiderato.
- **Postcondition:** L'utente ha aggiunto il brano ad una playlist, perciò selezionandola potrà navigare nell'elenco e decidere di scaricarlo **(UC7)** o mettere like **(UC8)**.
- **Base Sequence:**
  - **1.** L'utente seleziona il brano desiderato (da una playlist, dalla ricerca, ecc).
  - **2.** L'utente clicca sul pulsante apposito.
  - **3.** L'utente seleziona la playlist alla quale aggiungere il brano.
  - **4.** L'utente ha correttamente inserito il brano nella playlist.
- **Exception Sequence:** Non è stata creata alcuna playlist dall'utente o connessione fallita.
  - Nel caso non sia stata creata ancora nessuna playlist si ritorna al passo 3 del presente caso d'uso.
  - Nel caso di connessione fallita si ritorna al passo 1 della Base Sequence del caso d'uso **UC1**.

3.2.10 UC15 – Visualizza playlist

Il presente caso d'uso consiste nel visualizzare i brano contenuti in una delle proprie playlist.

- **Actor:** Utente base.
- **Precondition:** L'utente deve aver effettuato il log in e deve aver creato almeno una playlist.

- **Postcondition:** L'utente può consultare il contenuto della playlist, perciò potrà navigare nell'elenco e decidere di scaricare un brano selezionato **(UC7)** o mettere like **(UC8)**.
- **Base Sequence:**
  - **1.** L'utente seleziona l'elenco delle proprie playlist dal menu.
  - **2.** L'utente clicca sulla playlist.
  - **3.** L'utente ha correttamente eseguito l'operazione: può navigare fra i brani contenuti nella playlist.
- **Exception Sequence:** Non è stata creata alcuna playlist dall'utente o connessione fallita.
  - Nel caso non sia stata creata ancora nessuna playlist si ritorna al passo 1 del presente caso d'uso.
  - Nel caso di connessione fallita si ritorna al passo 1 della Base Sequence del caso d'uso **UC1**.

3.2.11 UC16 – Crea nuova playlist

Il presente caso d'uso consiste nel creare una nuova playlist al fine di inserire brani al suo interno.

- **Actor:** Utente base.
- **Precondition:** L'utente deve aver effettuato il log in.
- **Postcondition:** L'utente può aggiungere brani alla playlist appena creata **(UC9)**
- **Base Sequence:**
  - **1.** L'utente seleziona l'elenco delle proprie playlist dal menu.
  - **2.** L'utente clicca sul "Crea nuova playlist".
  - **3.** L'utente ha correttamente eseguito l'operazione: può aggiungere i brani alla playlist.
- **Exception Sequence:** Connessione fallita.

- Nel caso di connessione fallita si ritorna al passo 1 della Base Sequence del caso d'uso **UC1**.

3.2.12 UC17 – Elimina playlist

Il presente caso d'uso consiste nell'eliminare una delle proprie playlist.

- **Actor:** Utente base.
- **Precondition:** L'utente deve aver effettuato il log in e deve aver creato almeno una playlist.
- **Postcondition:** L'utente ha eliminato la playlist, quindi non sarà più visibile nell'elenco delle proprie playlist.
- **Base Sequence:**
  - **1.** L'utente seleziona la playlist desiderata dall'elenco delle proprie playlist.
  - **2.** L'utente clicca su "Elimina playlist".
  - **3.** L'utente ha correttamente eliminato la playlist: essa non sarà più visibile nell'elenco.
- **Exception Sequence:** Non è stata creata alcuna playlist dall'utente o connessione fallita.
  - Nel caso non sia stata creata ancora nessuna playlist si ritorna al passo 1 del presente caso d'uso.
  - Nel caso di connessione fallita si ritorna al passo 1 della Base Sequence del caso d'uso **UC1**.

3.2.13 UC18 – Modifica playlist

Il presente caso d'uso consiste nel poter modificare delle proprie playlist: può modificare il nome della stessa o rimuovere un brano da essa.

- **Actor:** Utente base.

- **Precondition:** L'utente deve aver effettuato il log in e deve aver creato almeno una playlist.
- **Postcondition:** L'utente ha modificato con successo la playlist, quindi sarà visualizzato il nuovo nome/i brani eliminati non saranno più nell'elenco.
- **Base Sequence:**
  - **1.** L'utente seleziona la playlist desiderata dall'elenco delle proprie playlist.
  - **2.** L'utente clicca su modifica playlist.
  - **3.** L'utente modifica il nome/elimina i brani desiderati.
  - **4.** L'utente ha modificato con successo la playlist.
- **Exception Sequence:** Non è stata creata alcuna playlist dall'utente o connessione fallita.
  - Nel caso non sia stata creata ancora nessuna playlist si ritorna al passo 1 del presente caso d'uso.
  - Nel caso di connessione fallita si ritorna al passo 1 della Base Sequence del caso d'uso **UC1**.

3.2.14 UC10 – Cerca Utente

Il presente caso d'uso consiste nel cercare un utente specifico dalla barra di ricerca tramite precise parole chiave, al fine di selezionare un amico da poter aggiungere come amico.

- **Actor:** Utente base.
- **Precondition:** L'utente deve aver effettuato il log in.
- **Postcondition:** L'utente ha effettuato con successo la ricerca: può aggiungere l'utente fra i propri amici **(UC11)**
- **Base Sequence:**
  - **1.** L'utente clicca sulla barra di ricerca apposita nel menu dell'interfaccia.

  - **2.** L'utente digita delle parole chiave per cercare l'utente, ovvero il suo nome e cognome. artista.
  - **3.** L'utente ha correttamente eseguito la ricerca: può navigare fra i risultati e selezionare l'utente desiderato.

- **Exception Sequence:** Non esiste un utente che corrisponde ai criteri di ricerca o connessione fallita.
  - Nel caso di ricerca fallita si ritorna al caso 2 della Base Sequence del presente caso d'uso.
  - Nel caso di connessione fallita si ritorna al passo 1 della Base Sequence del caso d'uso **UC1**.

3.2.15 UC11 – Aggiungi Utente

Il presente caso d'uso consiste nell'aggiungere un utente specifico nella propria lista di amici.

- **Actor:** Utene base.
- **Precondition:** L'utente deve aver effettuato il log in e aver effettuato correttamente la ricerca.
- **Postcondition:** L'utente ha aggiunto correttamente l'utente desiderato che sarà visibile nella propria lista di amici.
- **Base Sequence:**
  - **1.** L'utente, dopo aver effettuato la ricerca, clicca sull'utente desiderato.
  - **2.** L'utente clicca su Aggiungi
  - **3.** L'utente ha correttamente aggiunto l'amico.
- **Exception Sequence:** Connessione fallita.
  - Nel caso di connessione fallita si ritorna al passo 1 della Base Sequence del caso d'uso **UC1**.

3.2.16 UC12 – Visualizza informazioni profilo

Il presente caso d'uso consiste nel poter visualizzare le proprie informazioni nel profilo, come ad esempio nome, cognome, email.

- **Actor:** Utente base.
- **Precondition:** L'utente deve aver effettuato il log in.
- **Postcondition:** L'utente ha correttamente eseguito l'operazione: può modificare le proprie informazioni personali **(UC13)**.
- **Base Sequence:**
  - **1.** L'utente clicca su "Account" navigando il menu.
  - **2.** L'utente ha correttamente eseguito l'operazione.
- **Exception Sequence:** Connessione fallita.
  - Nel caso di connessione fallita si ritorna al passo 1 della Base Sequence del caso d'uso **UC1**.

3.2.17 UC13 – Modifica profilo

Il presente caso d'uso consiste nel poter modificare le informazioni del proprio profilo inserite in fase di registrazione. Il presente caso d'uso è attivo solo dal lato amministratore e non è ancora disponibile per gli utenti regolari, ma il suo funzionamento è stato comunque riepilogato di seguito.

- **Actor:** Utente base.
- **Precondition:** L'utente deve aver effettuato il log in e visitare la pagina relativa alle informazioni personali.
- **Postcondition:** L'utente ha correttamente modificato il proprio profilo: saranno visualizzate le informazioni personali aggiornate.
- **Base Sequence:**
  - **1.** L'utente, dopo aver visualizzato le info del proprio profilo, clicca su modifica.
  - **2.** L'utente effettua le modifiche desiderate.

- **3.** L'utente clicca su Salva modifiche.
- **4.** L'utente ha correttamente eseguito l'operazione.

- **Exception Sequence:** Connessione fallita.
  - Nel caso di connessione fallita si ritorna al passo 1 della Base Sequence del caso d'uso **UC1**.

3.2.18 UC14 – Elimina profilo

Il presente caso d'uso consiste nel poter eliminare il proprio profilo. Il presente caso d'uso è attivo solo dal lato amministratore e non è ancora disponibile per gli utenti regolari, ma il suo funzionamento è stato comunque riepilogato di seguito.

- **Actor:** Utente base, Artista.
- **Precondition:** L'utente deve aver effettuato il log in.
- **Postcondition:** L'utente ha correttamente eliminato il proprio profilo, può registrarsi nuovamente **(UC1)** per usufruire dei servizi offerti al sistema.
- **Base Sequence:**
  - **1.** L'utente, dopo aver visualizzato le info del proprio profilo, clicca su elimina profilo.
  - **2.** L'utente conferma la decisione.
  - **3.** L'utente ha correttamente eseguito l'operazione.
- **Exception Sequence:** Connessione fallita.
  - Nel caso di connessione fallita si ritorna al passo 1 della Base Sequence del caso d'uso **UC1**.

3.3 Documentazione API

Nel back-end sono stati realizzati gli endpoint visibili in tabella. Questi endpoint sono le fondamenta per il funzionamento base della comunicazione tra utente e sistema.

Metodo	Endpoint	Funzione
views.remove_song_from_playlist	playlists/<int:playlist_id>/remove_song/<int:song_id>	Visualizza brani piaciuti
views.song_detail	songs/<int:song_id>	Visualizza dettagli brano
views.like_song	song/like/<int:song_id>	Like al brano
views.unlike_song	song/unlike/<int:song_id>	Unlike Brano
views.album_detail	albums/<int:album_id>	Visualizza dettagli album
views.add_song_to_playlist	playlist/addsong/<int:song_id>	Aggiungi brano a playlist
views.add_friends	people/add/<int:ordinaryuser_id>	Aggiungi amico
views.remove_friends	friends/remove/<int:ordinaryuser_id>	Rimuovi amico
views.discover	discover	Algoritmo Discover

Tabella 3.1: API

3.4 UML Component Diagram

Dopo aver implementato i casi d'uso sopracitati e le relative interfacce, le API, la struttura del sistema risulta come segue in 2.3 e 3.2. Al fine di facilitare la lettura e comprensione del grafico, la rappresentazione del sistema è stata divisa in due.

Autenticazione, Account, Amici In figura sono presenti tutte le interfacce relative alle componenti di Autenticazione e Account, primi casi d'uso implementati.

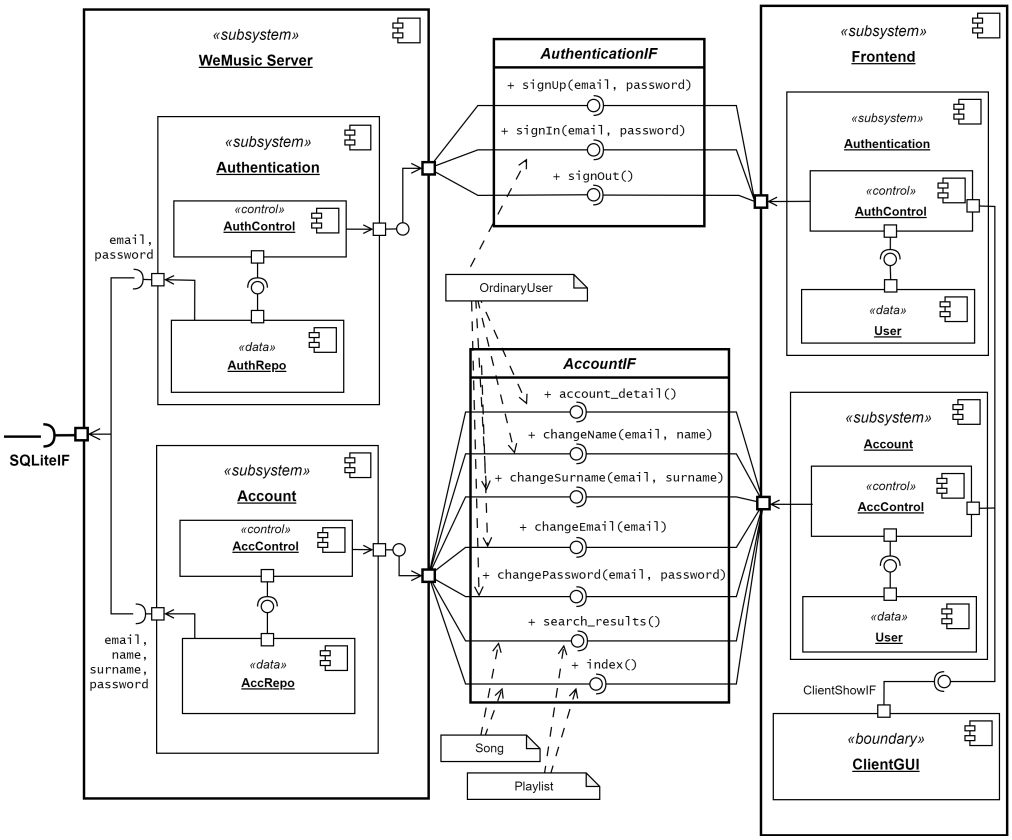

Figura 3.1: UML Component Diagram

Musica In figura sono presenti tutte le interfacce relative alla componente Musica, che include i brani, le playlist e gli album.

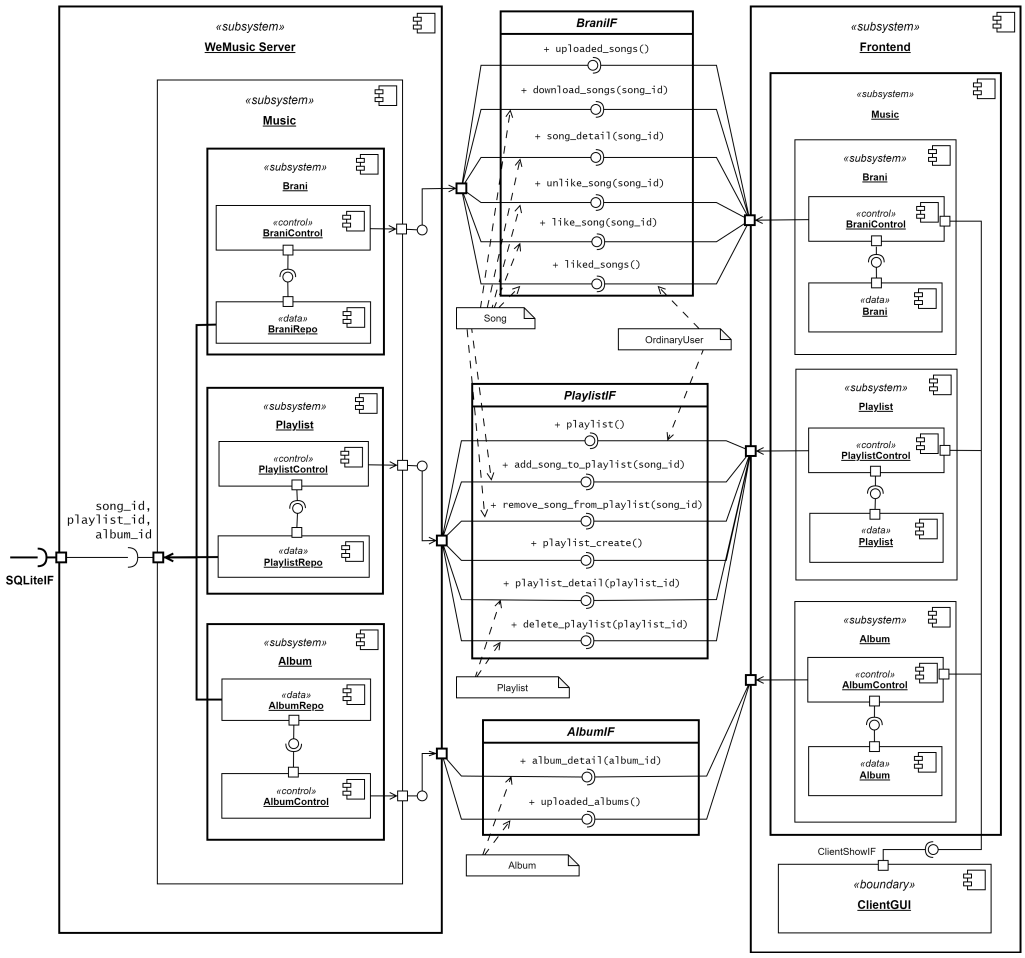

Figura 3.2: UML Component Diagram

Capitolo 4

Iterazione 3

4.1 Introduzione

Nella terza iterazione è stato approfondito e implementato l'algoritmo portante del sistema: l'algoritmo di Discover permette di "Scoprire" nuova musica e nuovi utenti in base alle preferenze dell'utente.

4.2 Implementazione del caso d'uso

4.2.1 UC19 – "Discover"

- **Actor:** Utente base.
- **Precondition:** L'utente deve aver effettuato il log in.
- **Postcondition:** L'utente può scoprire nuovi brani e aggiungere nuovi amici secondo i suggerimenti dell'algoritmo.
- **Base Sequence:**
  - **1.** L'utente, dopo aver effettuato il log in, clicca su Discover dal menu a sinistra.
  - **2.** L'utente naviga nella pagina dei brani e amici suggeriti.
  - **3.** L'utente ha correttamente eseguito l'operazione.
- **Exception Sequence:** Connessione fallita.
  - Nel caso di connessione fallita si ritorna al passo 1 della Base Sequence del caso d'uso **UC1**.

4.3 UML Component Diagram

In figura sono presenti tutte le interfacce relative alle componenti implementate in questa iterazione, ovvero Amici e Discover (parte relativa all'algoritmo).

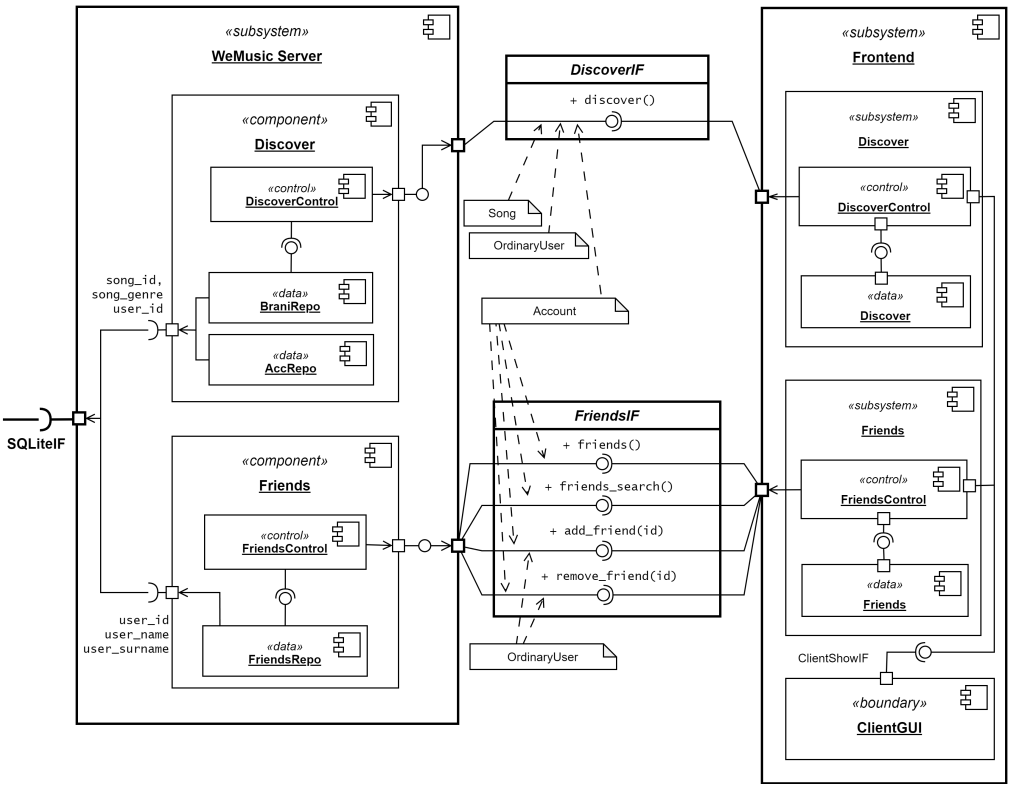

Figura 4.1: UML Component Diagram

4.4 Descrizione dell'algoritmo

L'algoritmo di Discover è stato ideato per offrire all'utente un'esperienza completa di riproduzione e scoperta di nuova musica, unendo a ciò la componente social: in questo modo si permette all'utente di conoscere nuove persone che condividono con lui i propri gusti musicali.

Questo tipo di algoritmo tiene traccia dei generi preferiti dell'utente e lo utilizza per suggerire musica simile, ma non solo: allo stesso modo riesce a suggerire una lista di utenti che hanno gusti simili all'utente loggato, in modo da poterli aggiungere alla propria lista di amici.

Le variabili e i fattori tenuti in considerazione per lo sviluppo dell'algoritmo di Discover sono i seguenti:

- Il genere favorito dell'utente, ottenuto da un'analisi dei suoi brani preferiti;
- L'età dell'utente;
- Il sesso dell'utente.

4.5 Funzionamento dell'algoritmo

Il funzionamento dell'algoritmo si basa su una previsione Machine Learning, tramite la libreria **Pandas** di Python; il nome fa riferimento sia a "Panel Data" che "Python Data Analysis", infatti è una libreria molto utilizzata per lavorare con dataset: tramite le sue molteplici funzioni consente di analizzare, pulire, esplorare e manipolare i dati, definendo conclusioni basate su teorie statistiche.

Come anticipato, in base al genere preferito dell'utente, il suo sesso e la sua età, l'algoritmo suggerisce dei brani ricavandoli da un dataset, filtrandoli secondo le variabili sopracitate. Al termine dell'esecuzione l'algoritmo mostrerà un elenco di brani e amici suggeriti.

Gli step di funzionamento dell'algoritmo possono essere così riassunti:

- Inizialmente, viene creato un oggetto account contenente le informazioni relative all'account dell'utente loggato, che ci serviranno in seguito; viene passato il parametro **request** che è quello che contiene le informazioni di interesse, ovvero l'età e il sesso (1=maschio, 0=femmina).

In seguito viene utilizzato un training set (**utenti.csv**) sul quale allenare la stima, contenente le informazioni degli utenti necessarie per estrapolare le variabili: quelle di input includono il sesso e l'età dell'utente, quella di output contiene solo il genere preferito (ciò che suggerirà l'algoritmo).

- Dopo aver creato queste due tabelle viene istanziato il modello previsionale tramite il comando **DecisionTreeClassifier**, un comando della libreria Pandas che permette, dandogli in input le variabili considerate, di fittare il modello che le lega.
- Viene quindi fittato il modello tramite il comando **modello.fit(...)**, che si allenerà sulle variabili scelte (età e sesso), per restituire il risultato (genere preferito).
- In seguito verrà effettuata la predizione inserendo le informazioni relative all'utente loggato tramite la funzione **modello.predict**, ed infine viene memorizzata la previsione relativa solo alla variabile di genere musicale.

4.5.1 Discover: Suggerimento brano

Nella parte successiva dell'algoritmo Discover si effettua un semplice filtro su tutte le canzoni presenti nel database del sistema e vengono selezionate solo quelle che rispettano il genere target restituito dall'algoritmo Discover; vengono in seguito salvate in una lista e proposte all'utente, che sarà in grado di navigare e selezionare quale scaricare o aggiungere ai preferiti o ad una playlist.

4.5.2 Discover: Suggerimento amici

Nell'ultima parte dell'algoritmo Discover viene seguito un processo più meccanico: facendo ancora riferimento al genere target individuato nella parte iniziale dell'algoritmo, viene estrapolata una lista di amici suggeriti per l'utente.

Inizialmente viene creata una lista amici vuota, vengono salvate le informazioni relative dell'utente loggato, vengono quindi selezionati tutti gli utenti presenti nella lista amici (in modo da escluderli in seguito nell'algoritmo). Dopo aver creato una lista degli identificativi, tramite un ciclo for each vengono selezionati gli utenti nel database sfogliando per ognuno le canzoni preferite, al fine di salvare il genere di ogni canzone in una variabile y.

Condizione per rientrare negli utenti suggeriti:

- L'utente analizzato (nel database) non deve essere l'utente loggato, ovvero devono avere id diversi;
- L'utente analizzato (nel database) non deve essere amico dell'utente loggato, ovvero il suo id non deve essere presente nella lista amici;
- All'utente analizzato (nel database) deve piacere almeno una canzone che abbia lo stesso genere target dell'algoritmo.

Nel context viene poi passata la lista creata con gli utenti selezionati dal database, in modo da permettere all'utente di poter navigare fra gli utenti ed, eventualmente, aggiungerli fra gli amici.

4.6 Studio della complessità

4.6.1 Algortimo Parte I

Pseudocodice Pseudocodice della prima parte dell'algoritmo, relativa all'inizializzazione e all'individuazione del genere target.

Algorithm 1: Step 1 - Inizializzazione

algoritmo *Discover(curr_gender, curr_age)* **begin**

/* Creazione oggetto account */
do $account \leftarrow request.account$

/* Memorizzazione utente corrente */
do $curr\_user \leftarrow curr\_user.\underline{get}()$

/* Lettura training set degli utenti.csv */
do $utenti \leftarrow read\_csv('utenti.csv')$

/* Creazione colonna di input della previsione */
do $X \leftarrow utenti.\underline{drop}(columns = ['genmusic'])$

/* Creazione colonna di output della previsione */
do $y \leftarrow utenti['genmusic']$

/* Creazione istanza del modello */
do $modello \leftarrow \underline{DecisionTreeClassifier}()$

/* Allenamento del modello sui dati */
do $modello.\underline{fit}(X.values, y.values)$

/* Previsione dei generi */
do $previsione \leftarrow modello.\underline{predict}(curr\_gender, curr\_age)$

/* Memorizzazione del genere target */
do $genere\_target \leftarrow previsione[0]$

...

Result: Genere target

end

UML Activity Diagram Diagramma delle attività della prima parte dell'algoritmo.

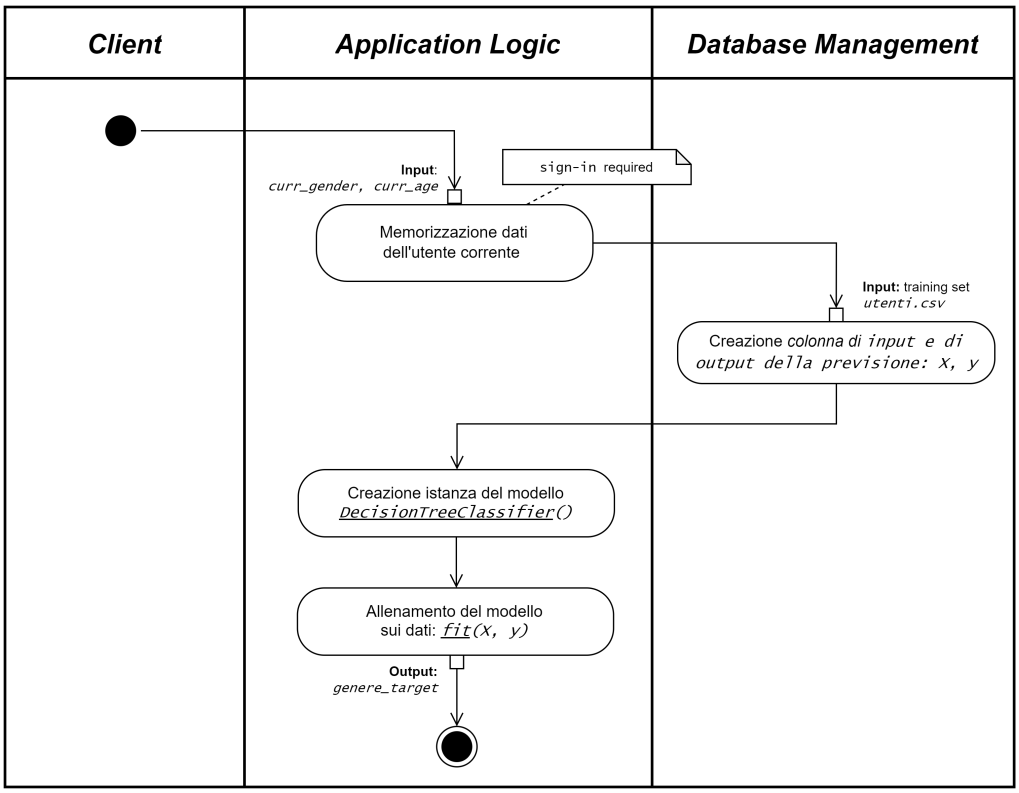

Figura 4.2: UML Activity Diagram (parte I)

4.6.2 Algortimo Parte II

Pseudocodice Pseudocodice della seconda parte dell'algoritmo, relativa al suggerimento dei brani in base al genere target.

Algorithm 2: Step 2 - Brani

```
algoritmo Discover(curr_gender, curr_age) begin
    ...
    /* Estrazione dei brani con genere target */
    do songs_list_db ← Songs.filter(genere = genere_target)

    /* Creazione lista dei brani che piacciono all'utente */
    do liked_songs_list ← curr_user.liked_song.values()

    /* Creazione lista vuota per gli id */
    do liked_songs_list_id ← []

    /* Inserimento degli ID nella lista */
    for sl in liked_songs_list do
        liked_songs_list_id.append(sl[id])
    end

    /* Creazione lista per i brani da suggerire all'utente*/
    do songs_list ← []

    /* Creazione lista vuota per i brani da suggerire
     all'utente*/
    for s in songs_list_db do
        if s.id not in liked_songs_list_id then
            songs_list.append(s)
        end
    end
    return songs_list
    ...
    Result: Lista brani con genere target
end
```

UML Activity Diagram Diagramma delle attività della seconda parte dell'algoritmo.

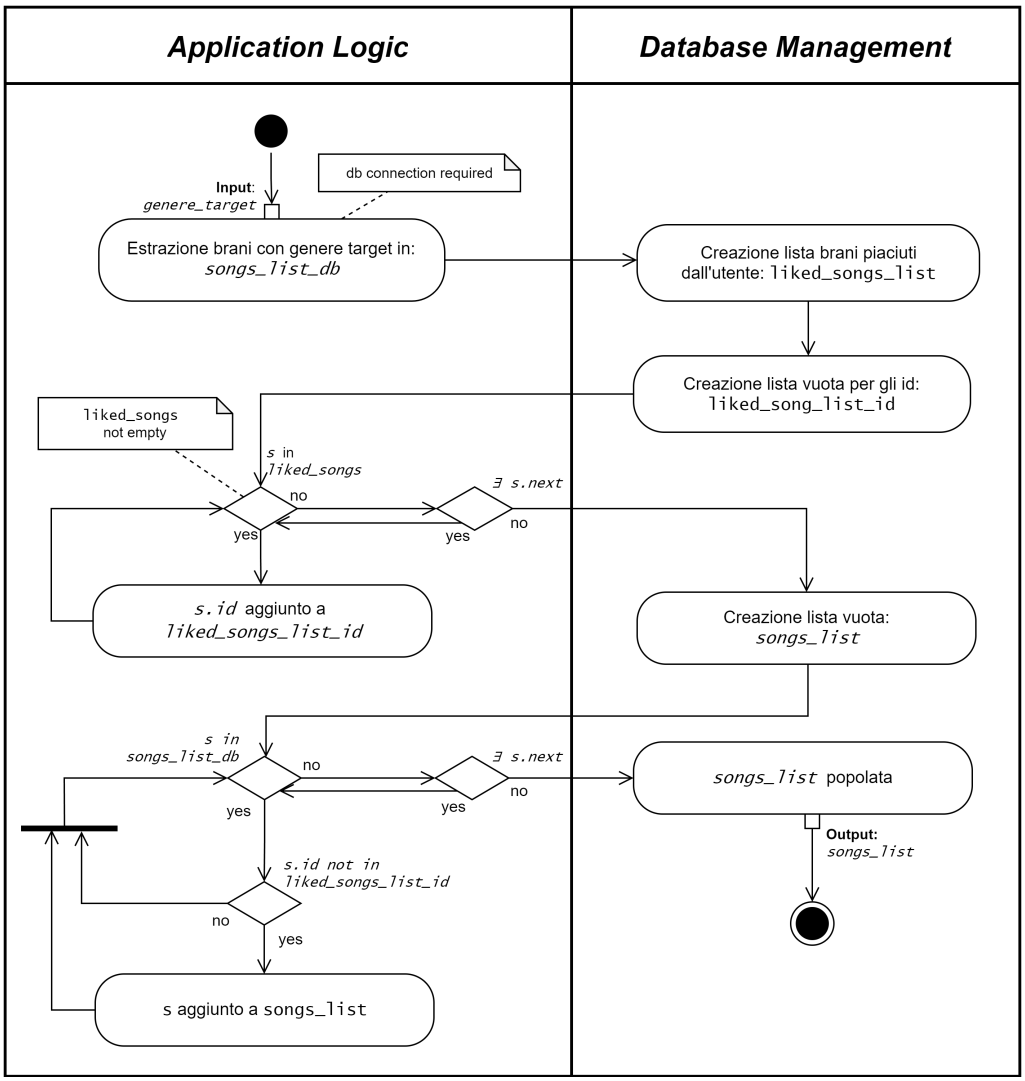

Figura 4.3: UML Activity Diagram (parte II)

4.6.3 Algoritmo Parte III

Pseudocodice Pseudocodice della terza parte dell'algoritmo, quella relativa al suggerimento di utenti con gusti simili.

Algorithm 3: Step 3 - Amici

```
algoritmo Discover(curr_gender, curr_age) begin
    ...
    /* Creazione lista vuota */
    do [] ← list_suggested_friends

    /* Memorizzazione della lista di amici dell'utente*/
    do friends_id ← curr_user.friends.values()

    /* Creazione lista vuota */
    do [] ← list_user_friends

    for v in friends_id do
        list_user_friends.append(v[id])
    end
    /* Memorizzazione amici dell'utente corrente */
    foreach x in user.all() do
        foreach y in x.liked_songs.all().values() do
            genere_temp ← y[genere]
            if genere_target = genere_temp AND x.account.id !=
             account.id AND x.account.id not in list_user_friends
             then
                list_suggested_friends.append(x)
            end
        end
    end
    return list_suggested_friends
    ...
    Result: Lista utenti con gusti affini all'utente
end
```

UML Activity Diagram Diagramma delle attività della terza parte dell'algoritmo.

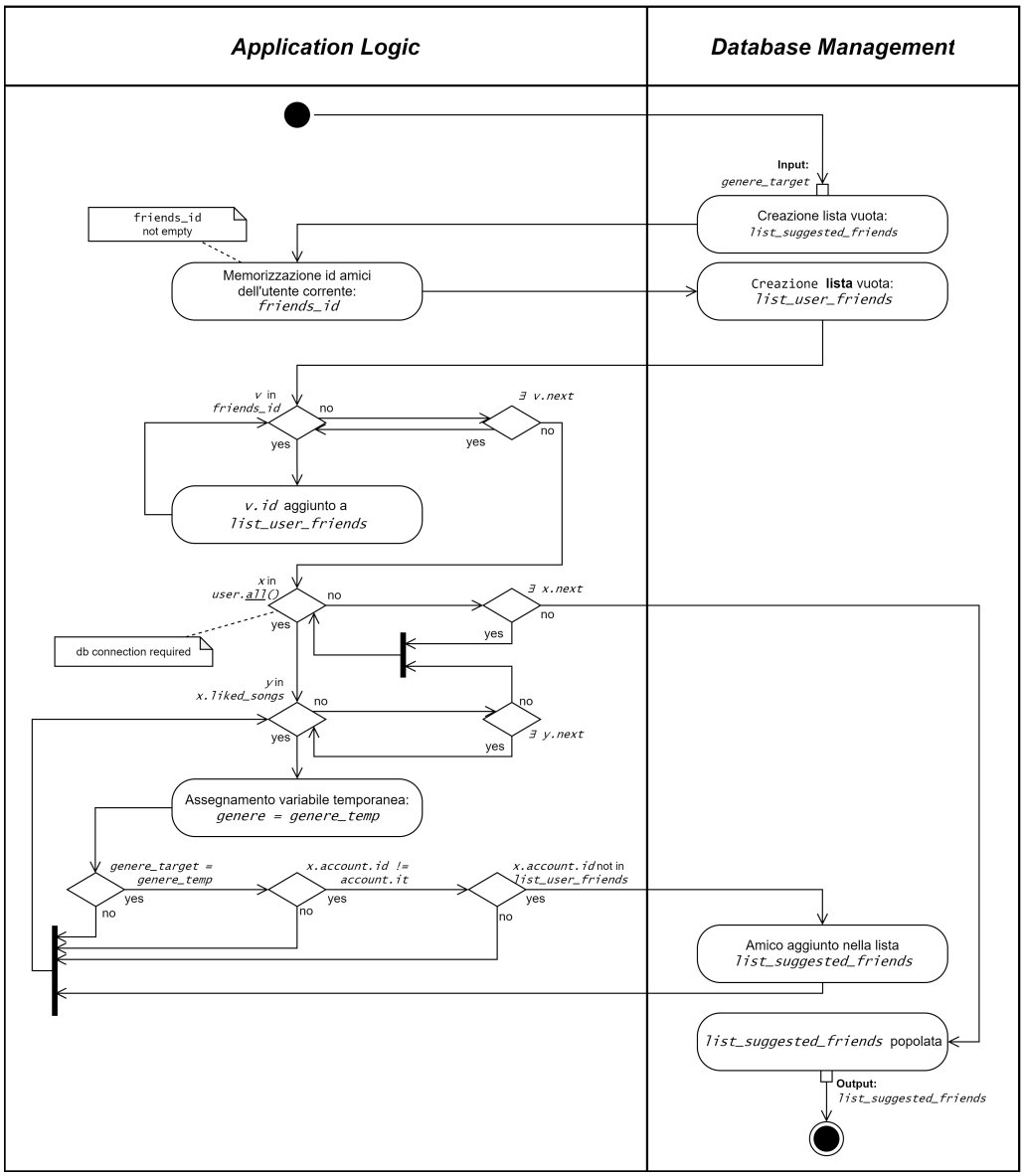

Figura 4.4: UML Activity Diagram (parte III)

4.6.4 Algoritmo Parte IV

Pseudocodice Pseudocodice dell'ultima parte dell'algoritmo.

Algorithm 4: Step 4 - Parametri di ritordno

algoritmo *Discover(curr_gender, curr_age)* **begin**

...

/* Ritorno parametri di interesse */
return *songs_list*
return *list_suggested_friends*

Result: Lista brani con genere target e lista utenti con gusti affini all'utente

end

UML Activity Diagram Diagramma delle attività della parte finale dell'algoritmo.

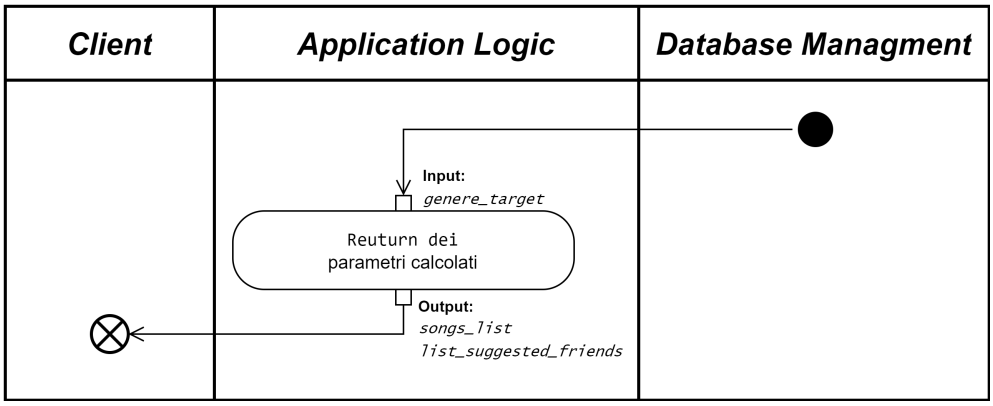

Figura 4.5: UML Activity Diagram (parte IV)

4.6.5 Analisi della complessità

Per valutare la complessità temporale dell'algoritmo, vengono esaminate le operazioni principali che vengono eseguite e e viene analizzato come queste operazioni dipendono dalla dimensione dell'input. Di seguito viene fornita una stima approssimativa della complessità temporale per ogni parte dell'algoritmo:

- **Lettura del file CSV:** La lettura del file CSV tramite la funzione *read_csv* ha una complessità temporale che dipende dalla dimensione del file, nello specifico dal numero di righe e colonne del file. Supponiamo che ci siano n righe e m colonne nel file. La lettura del file richiede un'operazione per ogni cella nel file, di conseguenza la complessità è dell'ordine di $O(n * m)$.

- **Creazione e addestramento del modello di classificazione:** La creazione del modello *DecisionTreeClassifier* e l'addestramento del modello tramite il metodo *fit* dipendono dal numero di caratteristiche e dal numero di esempi nel set di dati. Se denotiamo con p il numero di istanze (quindi n, le righe del file csv) e con q il numero di attributi (m, il numero di colonne), allora la complessità temporale può essere approssimata a $O(p * qlog(p))$, poiché la costruzione di un albero decisionale richiede operazioni che dipendono da entrambi i valori.

- **Previsione dei generi musicali:** La previsione dei generi musicali per l'utente corrente utilizzando il modello addestrato ha una complessità temporale di $O(1)$, poiché si tratta di un singolo esempio (operazione costante).

- **Filtraggio dei brani musicali:** Per ogni brano che piace all'utente corrente, viene estratto il relativo ID e memorizzato in una lista. Questa operazione richiede un'iterazione su tutti i s brani preferiti, quindi la complessità è $O(s)$. Il filtro dei brani musicali nel database attraverso la chiamata a *Song.objects.filter* può dipendere dal numero di brani nel database e dalla complessità della query. Questa operazione ha una complessità temporale dell'ordine di $O(s)$, dove s rappresenta il numero di brani che soddisfano i criteri di filtro.

- **Creazione della lista dei brani suggeriti:** Per ogni brano del training set con il genere musicale dedotto dall'algoritmo, viene verificato se l'ID del brano non è presente nella lista dei brani preferiti dall'utente corrente. Poiché questa operazione richiede l'iterazione su tutti i z brani nel training set, la complessità è $O(z)$.

- **Ricerca degli amici suggeriti:** La ricerca degli amici suggeriti coinvolge *due cicli for nidificati*. Supponendo che ci siano x utenti nel database e y canzoni preferite per ogni utente, la complessità temporale di questa parte dell'algoritmo può essere approssimata a $O(x * y)$.
- **Creazione del contesto:** La creazione del contesto finale ha una complessità temporale trascurabile, poiché coinvolge solo l'assegnazione di variabili.

Quindi, sommando tutte queste parti, si può concludere che la complessità temporale complessiva dell'algoritmo è approssimativamente:

$$\boldsymbol{O(n * m) + O(n * mlog(m)) + O(s) + O(z) + O(x * y) + O(1)}$$

Analisi complessità di *DecisionTreeClassifier* Il DecisionTreeClassifier fa parte di una libreria Python per il Machine Learning. L'addestramento di un albero decisionale utilizzando DecisionTreeClassifier si basa su una serie di divisioni dei dati in base alle caratteristiche (features) dei campioni. La complessità principale dell'addestramento di un albero decisionale è determinata dalla fase di costruzione dell'albero stesso. Alberi decisionali sono metodi di learning supervisionato non parametrici usati per la classificazione e la regressione. L'obiettivo è di creare un modello che preveda il valore di una variabile target tramite apprendimento di semplici regole decisionali e dedotte dalle caratteristiche dei dati. Quindi un albero può essere visto come una costantr approssimazione a tratti.

La complessità dell'addestramento di un albero decisionale è influenzata principalmente da due fattori:

- Il numero di campioni nel set di addestramento (n): Più campioni ci sono, più operazioni di divisione e valutazione devono essere eseguite.
- Il numero di caratteristiche (features) dei campioni nel set di addestramento (m): Più caratteristiche ci sono, più divisioni devono essere valutate.

Analisi complessità di *fit* Il metodo fit in Pandas viene utilizzato per adattare un modello ai dati; in questo caso, consente di produrre un modello di regressione lineare, insieme alla funzione analizzata precedentemente.

In conclusione, la complessità dell'addestramento di un albero decisionale con la funzione DecisionTreeClassifier e la regressione tramite fit risulta $O(n* m * log(m))$.

4.7 Analisi statica

4.7.1 Backend

Il testing statico dei campi è una pratica comune nel processo di sviluppo del software per verificare che i dati immessi in determinati campi rispettino determinate regole o vincoli. Per l'analisi statica del back-end è stato usato pylint, un linter configurabile per il linguaggio Python. Esso mette a disposizione varie funzionalità:

- imposizione di *coding standard* e di *code style*;
- *error detection*, sia a livello sintattico sia a livello di tipi;
- *refactoring* per codice non usato e/o duplicato;
- integrazione con vari IDE al fine di fornire tutto ciò in tempo reale.

Pylint è configurato tramite un file che contiene tutti parametri e le impostazioni da noi scelti per diminuire la complessità del codice ed aumentarne la chiarezza. Il linting del progetto avviene tramite uno script e, oltre a warning ed errori, nell'output è presente anche un singolo numero (su una scala da 1 a 10) che valuta il codice. In particolare, alla fine di questa iterazione questo valore è pari a **9.18/10**

4.8 Analisi dinamica

4.8.1 Testing Backend

Durante la terza iterazione sono stati sviluppati dei test su specifici campi delle classi, organizzati nel file test.py. Per supportare l'esecuzione dei test, sono state create funzioni ausiliarie raggruppate nel file utility.py. Di seguito vengono descritti i casi di test per i campi chiave delle classi Account, Album e Playlist.

Test per la classe Album: La classe Album gestisce le informazioni riguardanti un album musicale. È presente un caso di test che verifica la validità della data di pubblicazione.

- **isValidReleaseDate(const char* releaseDate):** Questo test verifica che la data di pubblicazione immessa per l'album sia valida. Se la data non rispetta un formato o una logica valida, il test restituisce false, altrimenti restituisce true.

```
def test_album_data_pub_correct(self):
    time = timezone.now() + datetime.timedelta(days=30)
    future_album = Album(pub_date = time)
    self.assertIs(future_album.album_pub_correct(),
                                                False)
```

Listing 4.1: class AlbumModelTests(TestCase)

Test per la classe Playlist: La classe Playlist gestisce le informazioni di una playlist musicale. È presente un caso di test che verifica la validità del nome della playlist:

- **isValidPlaylistName(const char* playlistName):** Questo test verifica che il nome immesso durante la creazione della playlist sia valido. Se il nome non rispetta determinati criteri o restrizioni, il test restituisce false, altrimenti restituisce true.

```
def test_playlist_name_correct(self):
    playlist_name_empty = ""
    playlist_err = Playlist(name = playlist_name_empty)
    self.assertIs(playlist_err.playlist_name_correct(),
                                                    False)
```

Listing 4.2: PlaylistModelTests(TestCase)

Test per la classe Account: La classe Account gestisce le informazioni dell'utente, come il nome e l'email. Durante la fase di registrazione, sono presenti tre casi di test che verificano la validità dei dati immessi:

- **isNameNotEmpty(const char* name)** Questo test verifica che il nome immesso durante la registrazione non sia vuoto. Se il nome è vuoto, il test restituisce false, altrimenti restituisce true;

```
def test_name_account_empty(self):
    name_fake = ""
    account_fake = Account(name = name_fake)
    self.assertIs(account_fake.account_name_correct(),
                                                    False)
```

Listing 4.3: class AccountModelTests(TestCase)

- **isNameWithoutNumbers(const char* name):** Questo test verifica che il nome immesso durante la registrazione non contenga numeri. Se il nome contiene almeno un numero, il test restituisce false, altrimenti restituisce true.

```
def test_name_account_number(self):
    name_err = ``Luca9"
    account_err = Account(name = name_err)
    self.assertIs(account_err.account_name_correct(),
                                                    False)
```

Listing 4.4: class AccountModelTests(TestCase)

- **isValidEmail(const char* email):** Questo test verifica che l'email immessa durante la registrazione sia valida. Se l'email non rispetta un formato valido, il test restituisce false, altrimenti restituisce true.

```
def test_email_account_correct(self):
    email_fake = ``pulcinopiowemusic.it"
    account_fake = Account(email = email_fake)
    self.assertIs(account_fake.account_email_correct(),
                                                    False)
```

Listing 4.5: class AccountModelTests(TestCase)

4.8.2 Coverage

Coverage.py è uno strumento atto a misurare la copertura del codice per il linguaggio Python. È anche disponibile un output interattivo in HTML, il cui output per il codice di questa iterazione è visibile in figura 4.6.

Coverage report: 95%

coverage.py v7.2.7, created at 2023-06-28 10:31 +0200

Module	statements	missing	excluded	coverage ↓
wmapp__init__.py	0	0	0	100%
wmapp\admin.py	8	0	0	100%
wmapp\apps.py	4	0	0	100%
wmapp\forms.py	10	0	0	100%
wmapp\migrations\0001_initial.py	8	0	0	100%
wmapp\migrations\0002_alter_album_pub_date_alter_ordinaryuser_friends_and_more.py	5	0	0	100%
wmapp\migrations\0003_account_name_account_surname_alter_account_email_and_more.py	5	0	0	100%
wmapp\migrations\0004_song_file_alter_album_pub_date_and_more.py	6	0	0	100%
wmapp\migrations\0005_alter_album_pub_date_alter_playlist_creation_date.py	5	0	0	100%
wmapp\migrations\0006_alter_album_pub_date_alter_playlist_creation_date.py	5	0	0	100%
wmapp\migrations\0007_alter_album_pub_date_alter_playlist_creation_date.py	5	0	0	100%
wmapp\migrations\0008_alter_album_pub_date_alter_playlist_creation_date.py	5	0	0	100%
wmapp\migrations\0009_account_birth_date_alter_album_pub_date_and_more.py	5	0	0	100%
wmapp\migrations\0010_account_sex_alter_album_pub_date_and_more.py	5	0	0	100%
wmapp\migrations__init__.py	0	0	0	100%
wmapp\tests.py	104	0	0	100%
wmapp\urls.py	3	0	0	100%
wmapp\utility.py	11	0	0	100%
wmapp\views.py	9	0	269	100%
wmapp\models.py	104	15	0	86%
Total	**307**	**15**	**269**	**95%**

coverage.py v7.2.7, created at 2023-06-28 10:31 +0200

Figura 4.6: Coverage.py results

Capitolo 5

Manuale Utente

5.1 Manuale Utente

Per confezionare l'applicazione e renderla eseguibile come un'applicazione desktop è stato utilizzato Electron: come anticipato questo framework consente di creare applicazioni desktop multi-piattaforma utilizzando tecnologie web come HTML, CSS e JavaScript.

Utilizzando Electron è stato possibile integrare il tutto in un'unica applicazione eseguibile, pronta per essere utilizzata dagli utenti senza la necessità di installare ulteriori dipendenze o configurazioni complesse, completa e pronta all'uso, che può essere distribuita facilmente tra gli utenti.

Capitolo 6

Conclusioni

6.1 Sviluppi futuri

Alcuni dei casi d'uso previsti originariamente non sono ancora stati completamente implementati, come era stato inizialmente pianificato. Tuttavia, il processo di sviluppo è ancora in corso e l'obiettivo è raggiungere una copertura completa dei casi d'uso identificati.

Di seguito è riportata una tabella riassuntiva che elenca i casi d'uso già implementati e quelli ancora da implementare nelle future fasi di sviluppo. Questo elenco mira a fornire una panoramica chiara delle attività svolte e delle attività rimanenti, consentendo una migliore pianificazione anche grazie al tipo di priorità assegnato al caso d'uso in questione.

I casi d'uso che non sono stati implementati si riferiscono specificamente alle funzionalità che sono attive solo dal lato amministratore e non sono ancora disponibili per gli utenti regolari. Anche se queste funzionalità possono essere funzionanti e operative nel contesto amministrativo, non possono ancora essere considerate come completamente implementate fino a quando non saranno accessibili e utilizzabili anche dagli utenti standard.

Pertanto, è importante distinguere tra le funzionalità che sono attualmente disponibili solo per gli amministratori e quelle che sono pienamente accessibili e utilizzabili dagli utenti regolari.

6.1.1 Casi d'uso implementati

Codice	Caso d'uso	Priorità
UC2	Sign in	Alta
UC3	Sign out	Alta
UC4	Cerca Brano	Alta
UC5	Cerca Album	Media
UC6	Cerca Artista	Media
UC7	Scarica Brano	Alta
UC8	Like al brano	Alta
UC9	Aggiungi brano a Playlist	Alta
UC15	Visualizza Playlist	Alta
UC16	Crea nuova Playlist	Alta
UC17	Elimina Playlist	Alta
UC18	Modifica Playlist	Media
UC10	Cerca Utente	Alta
UC11	Aggiungi Utente	Alta
UC12	Visualizza informazioni profilo	Media
UC19	Discover	Alta

Tabella 6.1: Tabella UC implementati

6.1.2 Casi d'uso non implementati

Codice	Caso d'uso	Priorità
UC1	Sign up	Alta
UC13	Modifica profilo	Media
UC14	Elimina profilo	Alta
UC20	Crea pagina Artista	Alta
UC21	Visualizza pagina Artista	Alta
UC22	Aggiungi Brano	Alta
UC23	Aggiungi Album	Media
UC24	Personalizza pagina Artista	Media
UC25	Consulta anagrafica	Media

Tabella 6.2: Tabella UC non implementati

6.2 Toolchain e tecnologie utilizzate

La seguente sezione comprende tutte le scelte di librerie, framework e piattaforme che sono state effettuate per l'implementazione del progetto.

6.2.1 Modellazione

- **Diagrammi UML di casi d'uso, deployment, delle classi:** Star UML, strumento di modellazione UML che consente agli sviluppatori e agli analisti di creare modelli visivi delle loro applicazioni software; supporta anche l'importazione ed esportazione di file in vari formati UML, facilitando la collaborazione e lo scambio di modelli. con altri strumenti di modellazione UML.

- **Diagrammi UML di componenti, attività:** Draw.io, uno strumento versatile che può essere utilizzato sia online che offline ed è disponibile come open source. È utilizzato per creare una vasta gamma di diagrammi generici e, oltre a ciò, offre anche strumenti specifici per la modellazione UML.

6.2.2 Linguaggi e librerie

Backend

Per l'implementazione del back-end sono stati usati:

- **Python** come linguaggio di programmazione, scelto per la sua semplicità e portabilità;

- **PIP** come package manager per Python;

- **Django**, un framework Python per la realizzazione di server, gestione di database e templating;

- **SQLite3** come DBMS, un dialetto SQL che offre un'ottima integrazione con Python; pur non essendo il più performante, è stato scelto per rapidità di implementazione, e in iterazioni future è possibile scambiarlo con altri DBMS più performanti;

- **Pylint** per l'analisi statica del codice Python, l'identificazione realtime di errori, warning e la fornitura di ottimizzazioni e suggerimenti;

- **Django Test Framework** e **Coverage.py** per l'analisi dinamica del codice Python, dei quali il primo è integrato in Django, mentre il secondo è uno strumento esterno per misurare la copertura del codice che fornisce anche un output interattivo in HTML.

Frontend

Per l'implementazione del front-end sono stati usati:

- **Bootstrap**, un framework che comprende svariati stili di default per velocizzare lo sviluppo di una UI coerente e reattiva;
- **HTML** e **CSS** come linguaggi di markup e styling delle pagine web;
- **Electron**, una libreria usata per poter compilare il codice in una app cross-platform.

6.2.3 Documentazione, Versioning e Organizzazione del team

La documentazione è stata scritta in **Latex**, compilata tramite **Visual Studio Code**. Il sistema di versioning utilizzato è **Git**, un sistema di versioning distribuito, open source e gratuito. Il codice del progetto è presente anche su **GitHub**, un sito per remote hosting di progetti git. L'organizzazione del team è avvenuta tramite **Discord**.

6.2.4 Ambienti e IDEs

Lo strumento principale utilizzato è **Visual Studio Code**, un editor di testo open source e gratuito realizzato per i sistemi operativi Linux, macOS e Windows.

Esso non è un IDE, in quanto non offre funzionalità di build automation e debugging, ma si basa invece su un sistema di estensioni, sia per syntax highlighting, linting, testing e building, mentre tutti gli strumenti per il funzionamento dei linguaggi sono locali e forniti dal sistema operativo.

Fra le molte funzionalità, è presente anche la *condivisione live* del codice, in modo da poterne discutere in real-time e poter usare tecniche di sviluppo agili che richiedono più persone, come il *pair programming*, anche a distanza.

6.3 Analisi del sw

6.3.1 Requisiti non funzionali

Le tecnologie sopracitate nel paragrafo relativo alla Toolchain, offrono anche dei precisi requisiti non funzionali, ovvero dei requisiti di qualità che definiscono le caratteristiche e le proprietà di un sistema software che vanno oltre la sua funzionalità specifica.

Mentre i requisiti funzionali descrivono cosa fa il sistema, i requisiti non funzionali specificano come il sistema dovrebbe essere in termini di prestazioni, sicurezza, usabilità, affidabilità e altri aspetti.

Le tecnologie utilizzate quindi riescono ad offrire scalabilità, manutenibilità, prestazioni adeguate per applicazioni di piccole e medie dimensioni, portabilità multi-piattaforma, sicurezza avanzata e affidabilità grazie all'uso di framework consolidati e al supporto di una vasta comunità di sviluppatori.

Di seguito vengono analizzati più nel dettaglio:

- **Scalabilità** Django, essendo un framework web robusto e scalabile, offre funzionalità di scalabilità orizzontale e verticale. Può gestire un numero crescente di richieste e traffico senza compromettere le prestazioni. Inoltre, SQLite può essere sostituito con un database più scalabile come MySQL o PostgreSQL per gestire un carico più elevato.

- **Manutenibilità** Django segue il principio di "don't repeat yourself" (DRY) e promuove una struttura organizzata del codice attraverso le sue convenzioni. Ciò rende più facile la manutenzione e l'estensione dell'applicazione nel tempo. Inoltre, l'uso di Python come linguaggio di programmazione favorisce una sintassi pulita e leggibile, facilitando la comprensione e la manutenzione del codice.

- **Prestazioni** L'utilizzo di Django e Python per il backend, insieme a SQLite come database leggero, offre prestazioni efficienti per applicazioni di piccole e medie dimensioni. Tuttavia, per applicazioni ad alto carico o con requisiti di prestazioni particolarmente elevati, è possibile sostituire SQLite con un database più performante come MySQL o PostgreSQL.

- **Portabilità** Utilizzando Electron per creare un'applicazione desktop multi-piattaforma, è possibile rendere l'applicazione compatibile con diversi sistemi operativi come Windows, macOS e Linux. Ciò consente agli utenti di utilizzare l'applicazione indipendentemente dalla piattaforma scelta.

- **Sicurezza** Django è noto per le sue robuste funzionalità di sicurezza. Offre meccanismi di autenticazione e autorizzazione integrati, prevenzione di attacchi CSRF (Cross-Site Request Forgery) e protezione contro le vulnerabilità comuni come SQL injection e XSS (Cross-Site Scripting). Tuttavia, è importante adottare le best practice di sicurezza durante lo sviluppo e la configurazione dell'applicazione per garantire un livello adeguato di sicurezza.

- **Affidabilità** Django è ampiamente utilizzato e testato dalla comunità open source, il che contribuisce alla sua affidabilità. Inoltre, l'uso di tecnologie consolidate come Django, Bootstrap e SQLite riduce il rischio di problemi imprevisti e aiuta a fornire un'esperienza stabile agli utenti.

6.3.2 Design Pattern

- **Abstraction (Astrazione)** Django utilizza l'astrazione attraverso i modelli per rappresentare concetti del dominio e le interazioni con il database.

- **Encapsulation (Incapsulamento)** Python e Django promuovono l'incapsulamento attraverso l'uso di classi e metodi per raggruppare dati e funzionalità correlate.

- **Information Hiding (Nascondere le informazioni)** Questo principio è supportato da Python e Django attraverso l'uso di attributi privati e metodi che nascondono gli aspetti interni dell'implementazione.

- **Separation of concerns (Separazione delle responsabilità)** Django segue il principio di separazione delle responsabilità attraverso l'architettura MVC, separando il modello, la vista e il controller.

- **Separation of interface and implementation (Separazione dell'interfaccia e dell'implementazione)** Django supporta la separazione dell'interfaccia e dell'implementazione attraverso le view, che definiscono l'interfaccia utente, separate dal modello e dal controller.

- **Coupling and cohesion (Coupling e coesione)** Django promuove una bassa accoppiamento e alta coesione tra i componenti attraverso l'architettura MVC, il sistema di routing e l'organizzazione modulare delle applicazioni.

- **Single point of reference (Singolo punto di riferimento)** Django offre il concetto di URLconf, che funge da singolo punto di riferimento per le richieste HTTP e dirige il traffico verso le view appropriate.
- **Divide and conquer (Dividere e conquistare)** Questo principio di progettazione generale può essere applicato nel modo in cui viene organizzato il codice sia nel backend (Python e Django) che nel frontend (Bootstrap).