Progetto di Tecnologie Informatiche per il Web

Michele Sangaletti

Indice

1. Traccia

Versione HTML pura

Un'applicazione web consente la gestione di una playlist di brani musicali.

Playlist e brani sono personali di ogni utente e non condivisi. Ogni utente ha username, password, nome e cognome. Ogni brano musicale è memorizzato nella base di dati mediante un titolo, l'immagine e il titolo dell'album da cui il brano è tratto, il nome dell'interprete (singolo o gruppo) dell'album, l'anno di pubblicazione dell'album, il genere musicale (si supponga che i generi siano prefissati) e il file musicale. Non è richiesto di memorizzare l'ordine con cui i brani compaiono nell'album a cui appartengono. Si ipotizzi che un brano possa appartenere a un solo album (no compilation). L'utente, previo login, può creare brani mediante il caricamento dei dati relativi e raggrupparli in playlist. Una playlist è un insieme di brani scelti tra quelli caricati dallo stesso utente. Lo stesso brano può essere inserito in più playlist. Una playlist ha un titolo e una data di creazione ed è associata al suo creatore.

A seguito del LOGIN, l'utente accede all'HOME PAGE che presenta l'elenco delle proprie playlist, ordinate per data di creazione decrescente, un FORM per caricare un BRANO con tutti i dati relativi e un form per creare una nuova playlist.
Il FORM per la creazione di una nuova PLAYLIST mostra l'elenco dei brani dell'utente ordinati per ordine alfabetico crescente dell'autore o gruppo e per data crescente di pubblicazione dell'album a cui il brano appartiene. Tramite il form è possibile selezionare uno o più brani da includere.
Quando l'utente clicca su una playlist nell'HOME PAGE, appare la pagina PLAYLIST PAGE che contiene inizialmente una tabella di una riga e cinque colonne. Ogni cella contiene il titolo di un brano e l'immagine dell'album da cui proviene. I brani sono ordinati da sinistra a destra per ordine alfabetico crescente dell'autore o gruppo e per data crescente di pubblicazione dell'album a cui il brano appartiene. Se la playlist contiene più di cinque brani, sono disponibili comandi per vedere il precedente e successivo gruppo di brani. Se la pagina PLAYLIST mostra il primo gruppo e ne esistono altri successivi nell'ordinamento, compare a destra della riga il bottone SUCCESSIVI, che permette di vedere il gruppo successivo. Se la pagina PLAYLIST mostra l'ultimo gruppo e ne esistono altri precedenti nell'ordinamento, compare a sinistra della riga il bottone PRECEDENTI, che permette di vedere i cinque brani precedenti. Se la pagina PLAYLIST mostra un blocco ed esistono sia precedenti sia successivi, compare a destra della riga il bottone SUCCESSIVI e a sinistra il bottone PRECEDENTI.
La pagina PLAYLIST contiene anche un FORM che consente di selezionare e AGGIUNGERE uno o più BRANI alla playlist corrente, se non già presente nella playlist. Tale form presenta i brani da scegliere nello stesso modo del form usato per creare una playlist. A seguito dell'aggiunta di un brano alla playlist corrente, l'applicazione visualizza nuovamente la pagina a partire dal primo blocco della playlist.
Quando l'utente seleziona il titolo di un brano, la pagina PLAYER mostra tutti i dati del brano scelto e il player audio per la riproduzione del brano.

Versione con JavaScript

Si realizzi un'applicazione client server web che modifica le specifiche precedenti come segue:

- Dopo il login dell'utente, l'intera applicazione è realizzata con un'unica pagina;
- Ogni interazione dell'utente è gestita senza ricaricare completamente la pagina, ma produce l'invocazione asincrona del server e l'eventuale modifica del contenuto da aggiornare a seguito dell'evento;
- L'evento di visualizzazione del blocco precedente/successivo è gestito a lato client senza generare una richiesta al server;

- L'applicazione deve consentire all'utente di riordinare le playlist con un criterio personalizzato diverso da quello di default. Dalla HOME con un link associato a ogni playlist si accede a una finestra modale RIORDINO, che mostra la lista completa dei brani della playlist ordinati secondo il criterio corrente (personalizzato o di default). L'utente può trascinare il titolo di un brano nell'elenco e di collocarlo in una posizione diversa per realizzare l'ordinamento che desidera, senza invocare il server. Quando l'utente ha raggiunto l'ordinamento desiderato, usa un bottone «salva ordinamento», per memorizzare la sequenza sul server. Ai successivi accessi, l'ordinamento personalizzato è usato al posto di quello di default. Un brano aggiunto a una playlist con ordinamento personalizzato è inserito nell'ultima posizione.

2. Documentazione ver. html pura

2.a. Analisi requisiti dati

Playlist e brani sono personali di ogni utente e non condivisi. Ogni **utente** ha **username**, **password**, **nome** e **cognome**. Ogni **brano musicale** è memorizzato nella base di dati mediante un **titolo**, l'**immagine** e il **titolo dell'album** da cui il brano è tratto, il **nome dell'interprete** (singolo o gruppo) dell'album, l'**anno di pubblicazione dell'album**, il **genere musicale** (**si supponga che i generi siano prefissati**) e il **file musicale**. Non è richiesto di memorizzare l'ordine con cui i brani compaiono nell'album a cui appartengono. Si ipotizzi che **un brano possa appartenere a un solo album** (no compilation). L'utente, previo login, può creare brani mediante il caricamento dei dati relativi e raggrupparli in playlist. **Una playlist è un insieme di brani scelti tra quelli caricati dallo stesso utente**. **Lo stesso brano può essere inserito in più playlist**. Una **playlist** ha un **titolo** e una **data di creazione** ed **è associata al suo creatore**.

Legenda:

- **Entità**
- **Attributi**
- **Relazioni**.

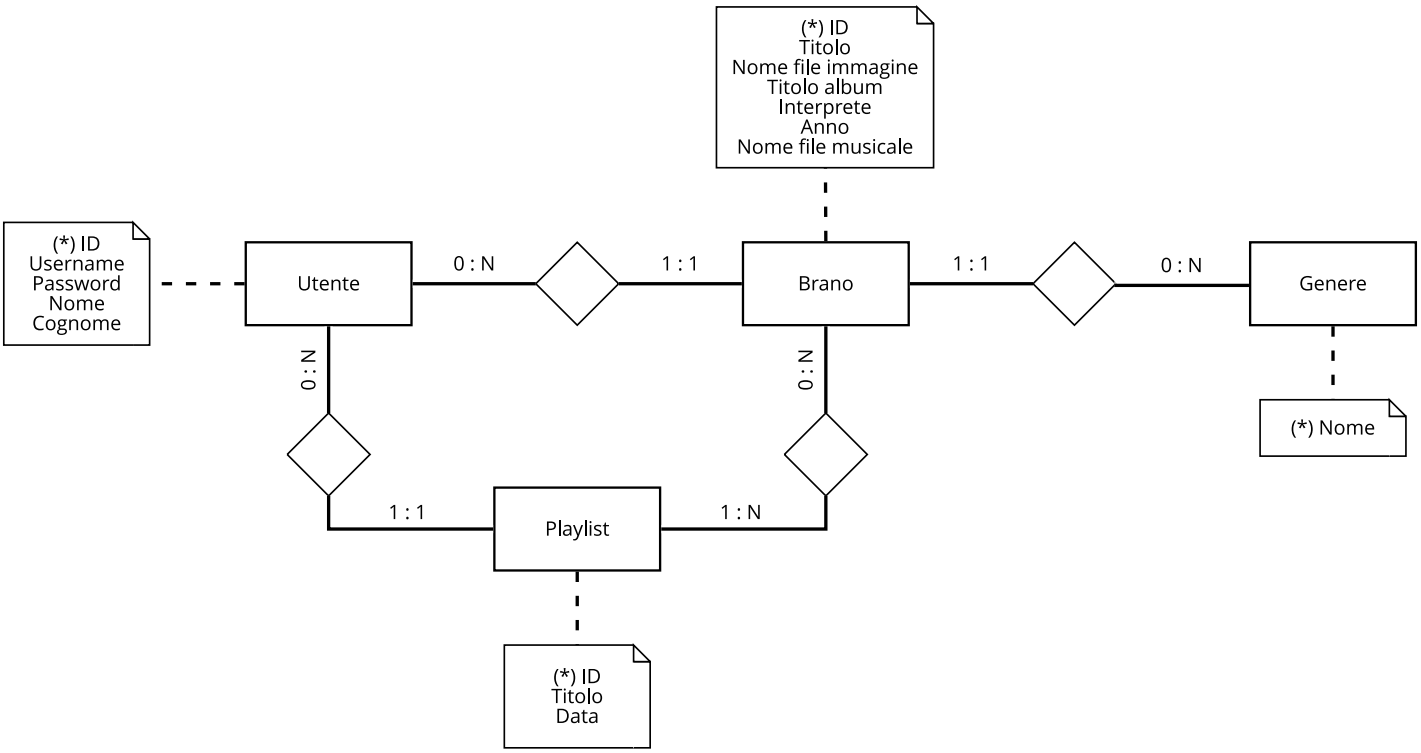


Diagramma entità-relazioni

```sql
create table users
(
    id       int auto_increment,
    username varchar(32) not null unique,
    password varchar(32) not null,
    name     varchar(32) not null,
    surname  varchar(32) not null,
    primary key (id)
);

create table genres
(
    name varchar(32),
    primary key (name)
);

create table songs
(
    id              int auto_increment,
    user_id         int          not null,
    title           varchar(256) not null,
    image_file_name varchar(256) not null,
    album_title     varchar(256) not null,
    performer       varchar(256) not null,
    year            int          not null check ( year > 0 ),
    genre           varchar(256) not null,
    music_file_name varchar(256) not null,
    primary key (id),
    foreign key (user_id) references users (id) on update cascade on delete no action,
    foreign key (genre) references genres (name) on update cascade on delete no action,
    unique (user_id, music_file_name),
    unique (user_id, title)
);

create table playlists
(
    id      int auto_increment,
    user_id int          not null,
    title   varchar(256) not null,
    date    date         not null default current_date,
    primary key (id),
    unique (user_id, title)
);

create table playlist_contents
(
    playlist int,
    song     int,
    primary key (playlist, song),
    foreign key (playlist) references playlists (id) on update cascade on delete no
action,
    foreign key (song) references songs (id) on update cascade on delete no action
);
```

Database design

2.b. Analisi requisiti d'applicazione

A seguito del **LOGIN**, l'utente accede all'**HOME PAGE** che presenta l'**elenco delle proprie playlist**, ordinate per data di creazione decrescente, un **FORM per caricare un BRANO con tutti i dati relativi** e **un FORM per creare una nuova playlist**.
Il FORM per la creazione di una nuova PLAYLIST mostra l'**elenco dei brani dell'utente ordinati per ordine alfabetico crescente dell'autore o gruppo e per data crescente di pubblicazione dell'album a cui il brano appartiene**. Tramite il form è possibile **selezionare uno o più brani da includere**.
Quando l'utente **clicca su una playlist nell'HOME PAGE**, **appare la pagina PLAYLIST PAGE** che contiene inizialmente una **tabella di una riga e cinque colonne. Ogni cella contiene il titolo di un brano e l'immagine dell'album** da cui proviene. **I brani sono ordinati da sinistra a destra per ordine alfabetico crescente dell'autore o gruppo e per data crescente di pubblicazione dell'album a cui il brano appartiene. Se la playlist contiene più di cinque brani, sono disponibili comandi per vedere il precedente e successivo gruppo di brani**. Se la pagina PLAYLIST mostra il primo gruppo e ne esistono altri successivi nell'ordinamento, **compare a destra della riga il bottone SUCCESSIVI, che permette di vedere il gruppo successivo**. Se la pagina PLAYLIST mostra l'ultimo gruppo e ne esistono altri precedenti nell'ordinamento, **compare a sinistra della riga il bottone PRECEDENTI, che permette di vedere i cinque brani precedenti**. Se la pagina PLAYLIST mostra un blocco ed esistono sia precedenti sia successivi, compare a destra della riga il bottone SUCCESSIVI e a sinistra il bottone PRECEDENTI.
La pagina PLAYLIST contiene anche un **FORM che consente di selezionare e AGGIUNGERE uno o più BRANI alla playlist corrente**, se non già presente nella playlist. Tale form **presenta i brani da scegliere nello stesso modo del form usato per creare una playlist**. **A seguito dell'aggiunta di un brano alla playlist corrente, l'applicazione visualizza nuovamente la pagina a partire dal primo blocco della playlist**.
Quando l'utente **seleziona il titolo di un brano**, la pagina **PLAYER** mostra tutti **i dati del brano scelto** e il **player audio** per la riproduzione del brano.

Legenda:

- **Pagine**;
- **Componenti**;
- **Eventi**;
- **Azioni**.

2.c. Aggiunta alle specifiche

- Funzione di logout, accessibile tramite un pulsante dalle pagine di home, playlist e canzone;
- Funzione per tornare alla homepage, accessibile tramite un pulsante dalle pagine di playlist e canzone;
- Funzione per tornare alla pagina della playlist originale, accessibile tramite un link dalla pagina della canzone;
- Messaggio di «benvenuto» quando l'utente è nella home page;
- Nella pagina della playlist viene mostrata la data in cui è stata creata;
- Possibilità di creare una playlist senza brani;
- Istruzioni sul creare una playlist nella homepage se l'utente non ha playlist associate;
- Istruzioni sull'aggiungere brani alla playlist se vuota nella pagina della playlist;
- Istruzioni sul caricare canzoni nella pagina della playlist se l'utente o ha aggiunto tutti i suoi brani alla playlist o non ha brani associati.

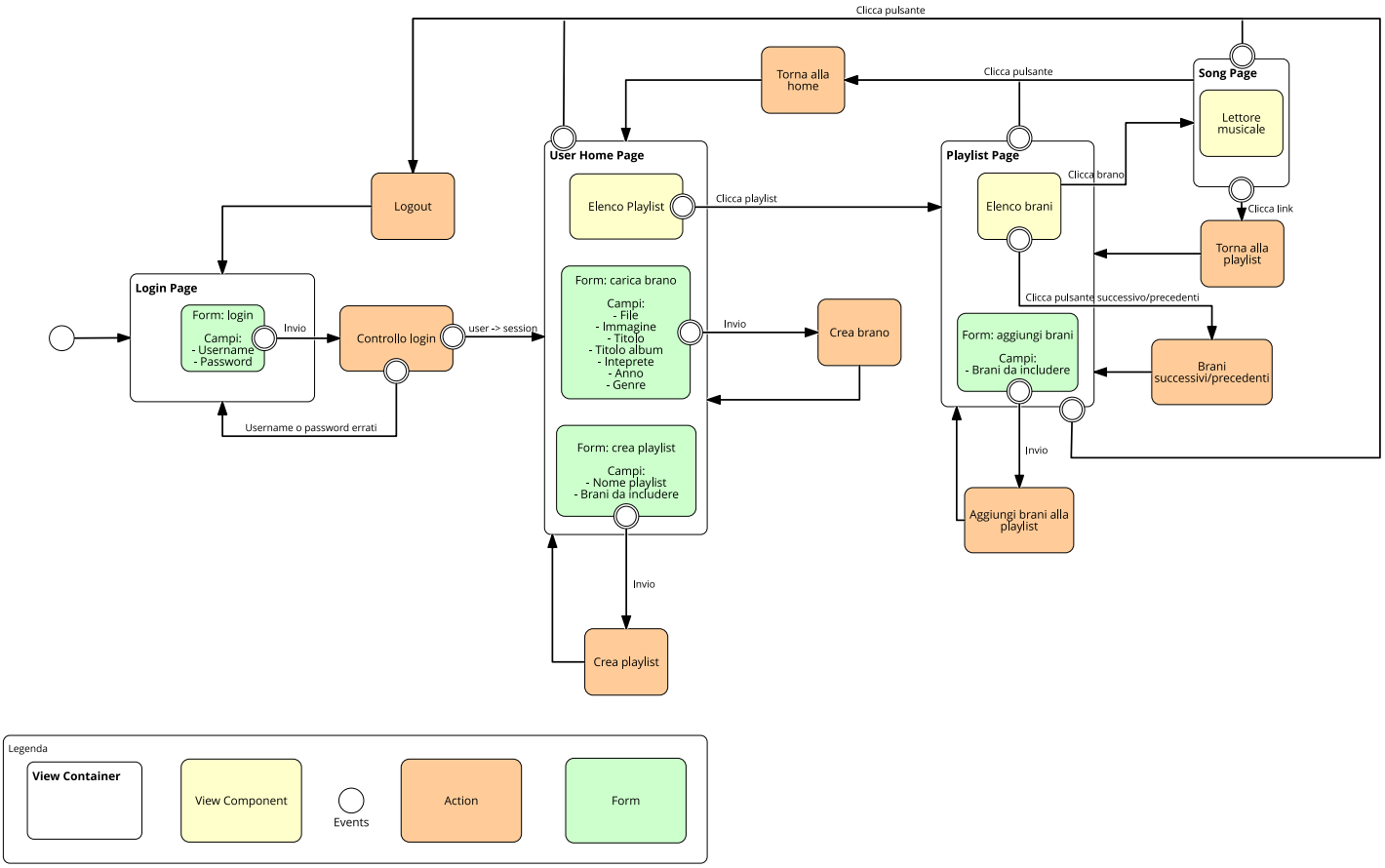


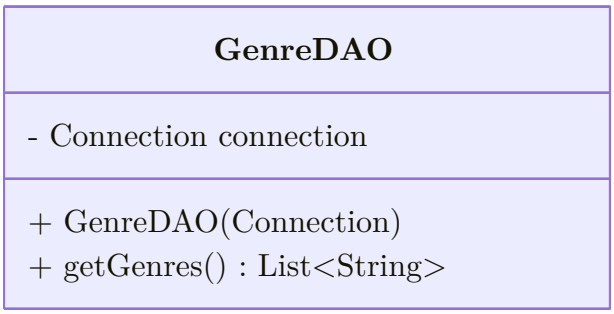
Diagramma IFML

2.d. Componenti e viste

2.d.a. Beans

Tutti gli attributi sono riconducibili al diagramma ER del database (tranne per la tabella `playlist_contents`, che è stata incorporata dentro l'oggetto `Playlist` per convenienza), mentre i metodi sono i soliti getter e setter di Java.

2.d.b. DAOs



- `getGenres()`: ritorna una lista dei generi come stringhe.

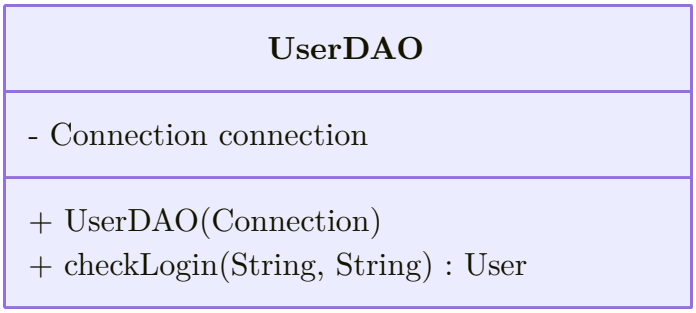
PlaylistDAO
- Connection connection
+ PlaylistDAO(Connection) + getFullPlaylist(int) : Playlist + addSongsToPlaylist(int, List<Integer>) : void + getPlaylistId(int, String) : int + getPlaylists(int) : List<Playlist> + insertPlaylist(int, String, List<Integer>) : void + getUserId(int) : int

- `getPlaylists(int userId)`: ritorna una lista di `Playlist` create dallo user associato al dato `userId`. Gli oggetti `Playlist` creati hanno l'attributo `songs = null`, dato che questa funzione viene chiamata solo per riempire la lista di playlist nella homepage;
- `getFullPlaylist(int playlistId)`: ritorna la `Playlist` con l'id dato con tutte le informazioni associate (compreso l'elenco di canzoni associate);
- `insertPlaylist(int userId, String title, List<Integer> songsId)`: crea una nuova playlist con le informazioni date e «oggi» come data di creazione;
- `addSongsToPlaylist(int playlistId, List<Integer> songsId)`: aggiunge la coppia (playlistId, songId) alla tabella `playlist_contents` per ogni id nella lista `songsId`;
- `getPlaylistId(int userId, String title)`: ritorna l'id della playlist che ha associati lo user id e il titolo passati;
- `getUserId(int playlistId)`: ritorna l'id dello user che ha creato la playlist, o -1 altrimenti.

SongDAO
- Connection connection
+ SongDAO(Connection) + getSongsIdFromUserId(int) : List<Integer> + getAllSongsFromPlaylist(int) : List<Song> + getSong(int) : Song + insertSong(int, String, String, String, String, int, String, String) : void - getSongFromResultSet(ResultSet) : Song + getAllSongsFromUserId(int) : List<Song> + getSongsNotInPlaylist(int, int) : List<Song> - getSongListFromResultSet(ResultSet) : List<Song>?

- `getAllSongsFromUserId(int userId)`: ritorna una lista di tutte le canzoni associate allo user dato, ordinate per interprete e anno, o `null` se non ne sono state trovate;
- `getAllSongsFromPlaylist(int playlistId)`: ritorna una lista di tutte le canzoni associate all'id della playlist dato, ordinate per interprete e anno, o `null` se non ne sono state trovate;

- `getSongListFromResultSet(ResultSet resultSet)`: metodo che estrare le canzoni dal set dato (se vuoto o senza canzoni, ritorna `null`);
- `insertSong(int userId, String title, String imageFileName, String albumTitle, String performer, int year, String genre, String musicFileName)`: aggiunge la canzone alla tabella `songs`;
- `getSong(int songId)`: ritorna un oggetto `Song` dato l'id della canzone;
- `getSongsNotInPlaylist(int userId, int playlistId)`: ritorna una lista di tutte le canzoni associate al dato user che non appartengono alla data playlist;
- `getSongsIdFromUserId(int userId)`: ritorna una lista di tutti gli id delle canzoni associate al dato user;
- `getSongFromResultSet(ResultSet resultSet)`: estrae un'oggetto `Song` dal dato resultSet.



- `checkLogin(String username, String password)`: controlla nel database se un utente con i dati username e password esiste: in caso affermativo, ritorna un oggetto `User` con le informazioni dell'utente, `null` altrimenti.

2.d.c. Controllers

- `AddSongs`;
- `CheckLogin`;
- `CreatePlaylist`;
- `GetFile`;
- `GoToHomepage`;
- `GoToPlaylist`;
- `GoToSong`;
- `Logout`;
- `UploadSong`.

2.d.d. Utils

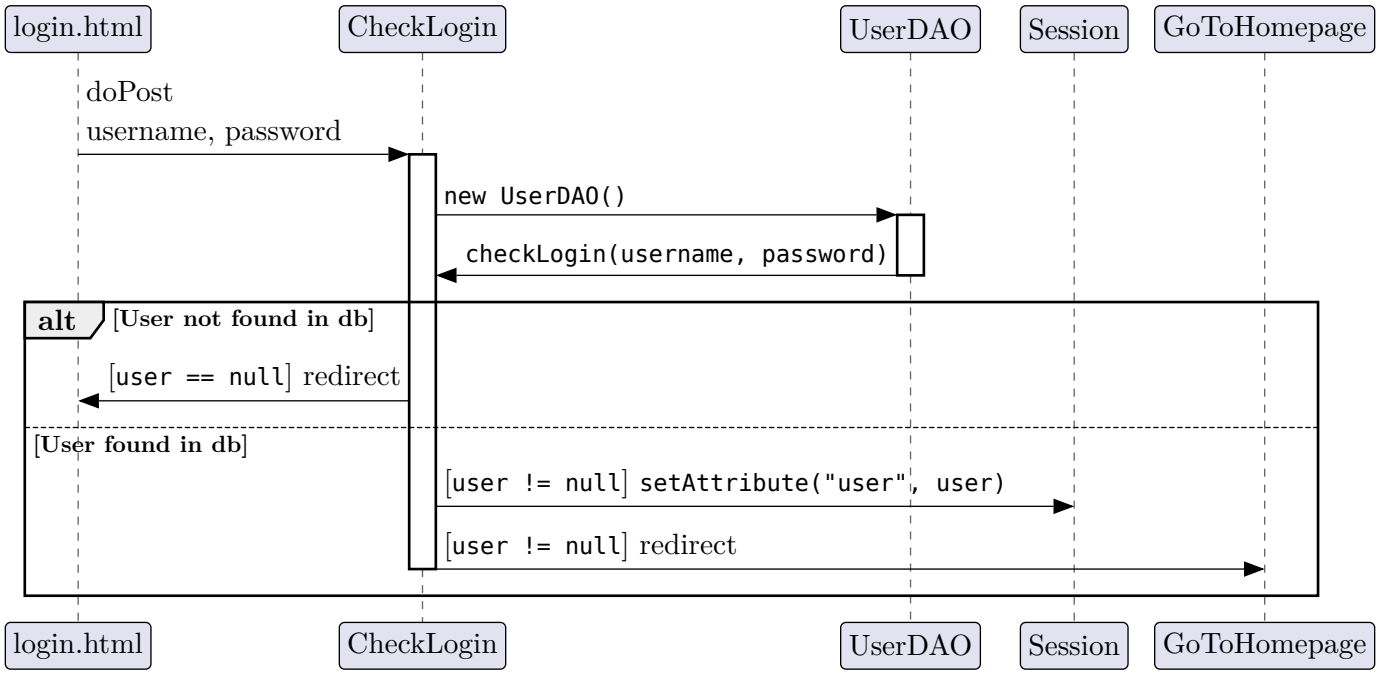
- `getConnection(ServletContext context)`: dato il contesto della servlet, inizializza e restituisce la connessione al database.

2.d.e. Templates

- `login.html` (welcome-file);
- `homepage.html`;
- `playlist.html`;
- `song.html`;

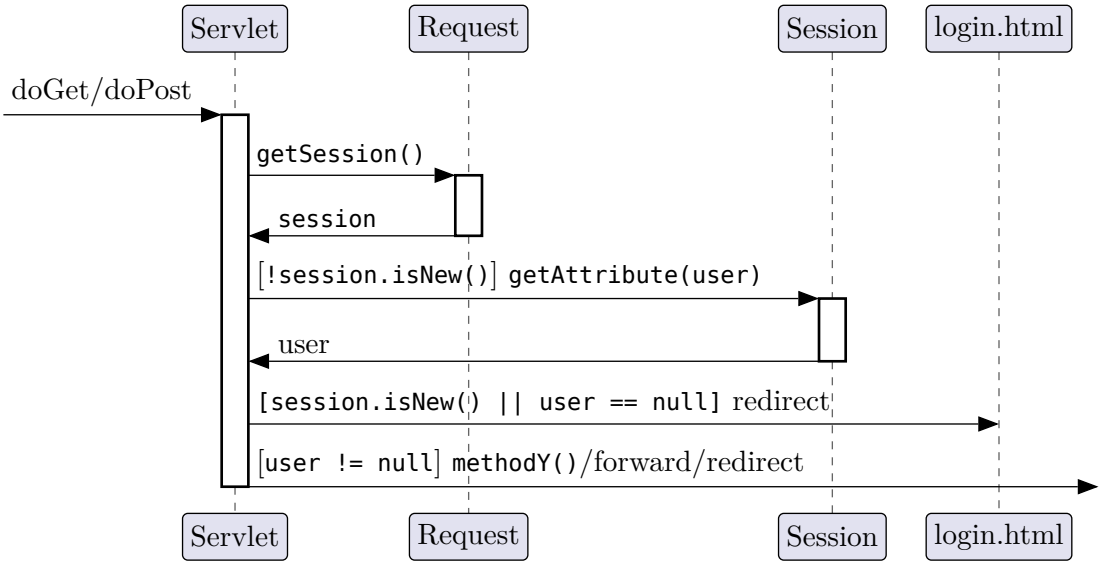
2.e. Sequence diagrams

Login



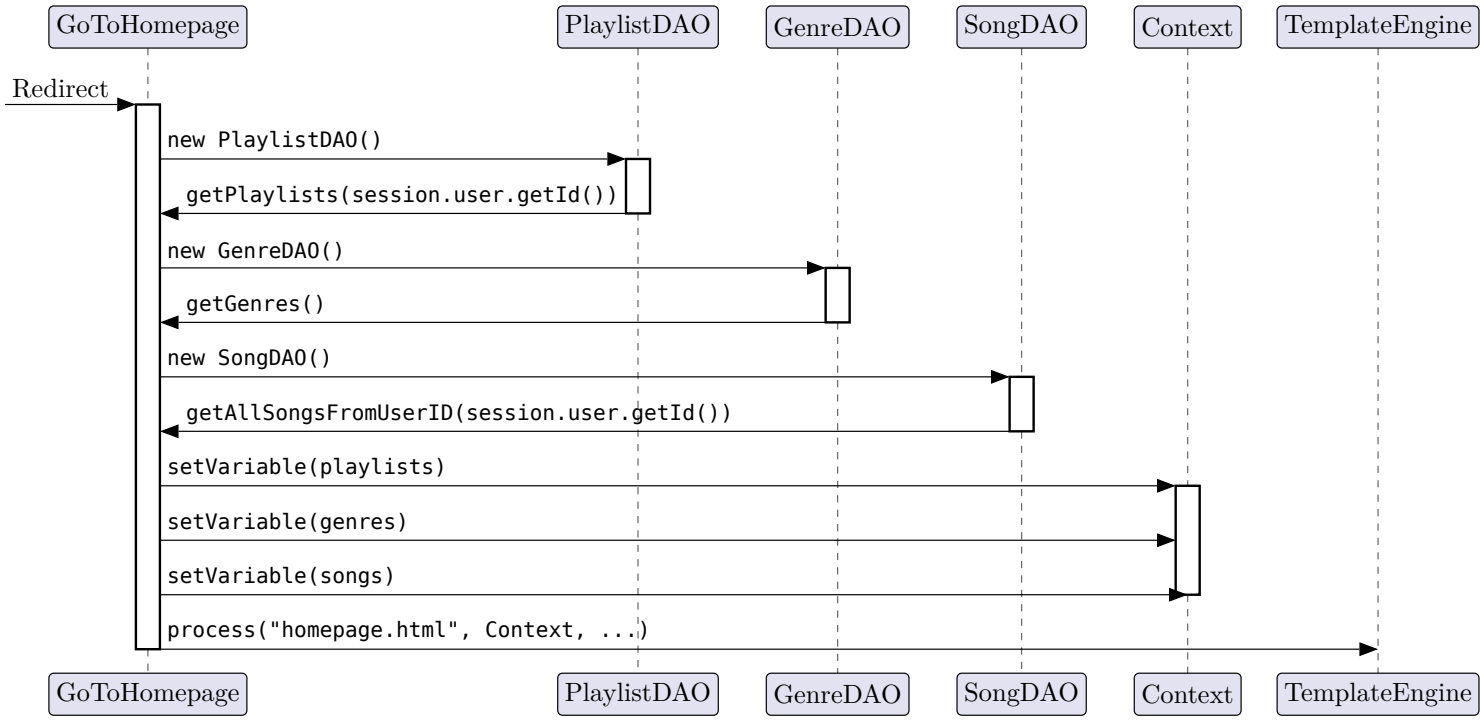
Dopo che l'utente ha inserito le credenziali nel form e l'ha inviato, la servlet `CheckLogin` controlla, tramite `UserDAO`, se un utente con quello username e quella password esiste nel database: in caso affermativo, salva l'oggetto `user` nella sessione e reindirizza l'utente verso `GoToHomepage`; se l'utente non viene trovato, il DAO ritorna `null` e viene ricaricata `login.html`.

Controllare l'user



Questo controllo viene fatto all'inizio di ogni servlet da qui in poi, ed è stato riportato separatamente per sintesi. Quando l'utente tenta di accedere a una servlet, questa controlla che la sessione non sia «nuova», e richiede l'attributo `user`. Se la sessione è nuova o l'`user` è `null`, l'utente viene indirizzato alla pagina di login, altrimenti la servlet procede.

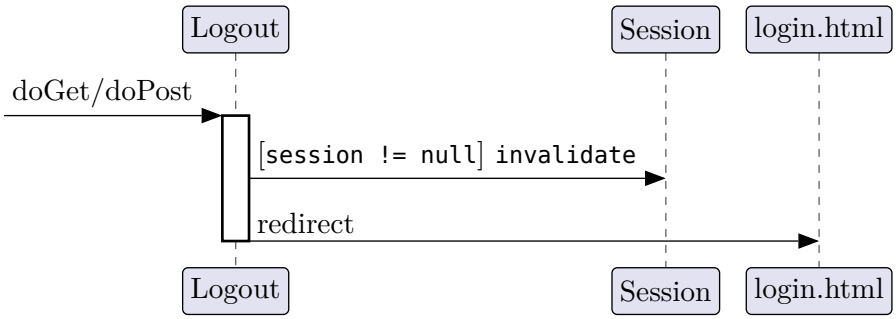
Tornare/andare alla homepage



Quando l'utente fa il login o ritorna alla homepage tramite l'apposito pulsante, viene chiamata la servlet `GoToHomepage`. Richiede la lista di playlist dell'utente dalla `PlaylistDAO`, i generi da `GenreDAO` e l'elenco delle canzoni dell'utente dal `SongDAO`. Questi vengono poi inseriti nel contesto e il template engine carica la pagina con le informazioni necessarie:

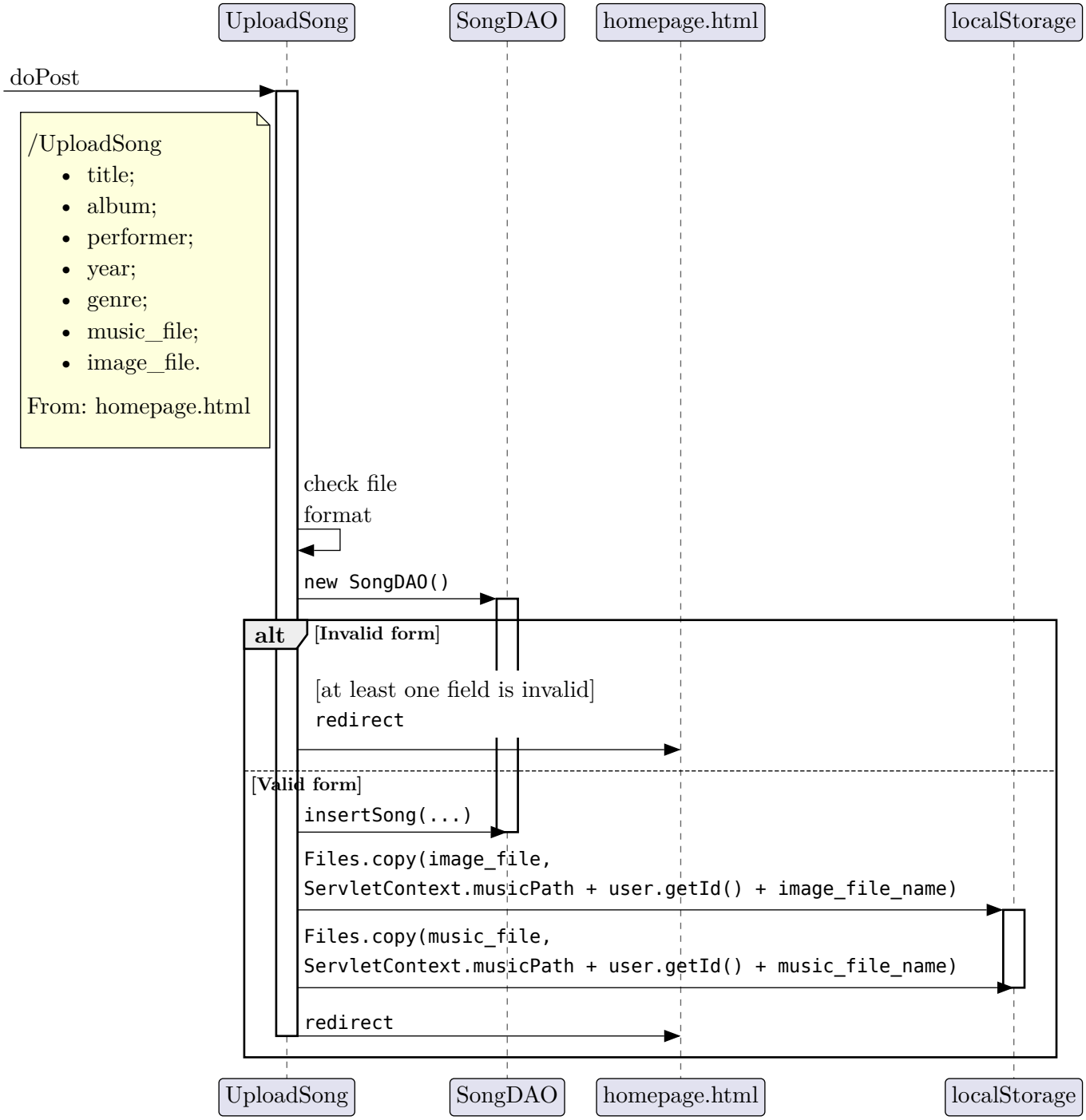
- Dalla sessione recupera lo user e mette il suo nome e cognome nel messaggio di benvenuto;
- Se la lista di playlist è vuota, viene renderizzato un messaggio che consiglia all'utente di usare il form apposito per creare una playlist. Se invece l'utente ha già creato delle playlist, viene renderizzata una tabella con il nome delle playlist (che serve da link per chiamare la servlet `GoToPlaylist`) e la sua data di creazione;
- I generi vengono caricati in un menu a tendina nel form apposito;
- Se la lista di canzoni è vuota, viene stampato un messaggio che consiglia all'utente di usare il form apposito per caricare un brano; altrimenti tutte le canzoni vengono renderizzate come opzioni per creare una nuova playlist. In entrambi i casi, viene renderizzata la casella di testo per inserire il nome della nuova playlist.

Logout



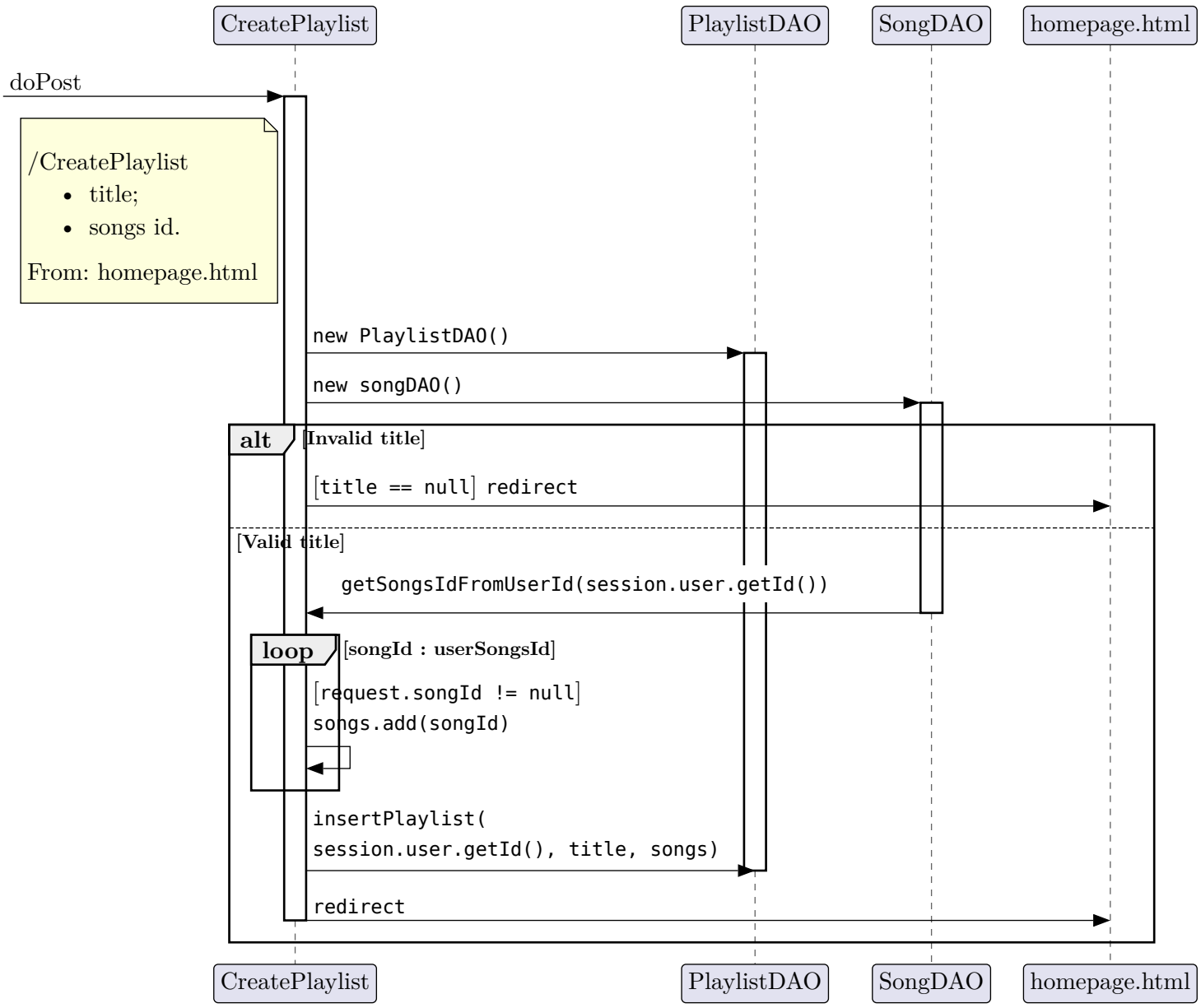
Quando l'utente clicca il pulsante di logout, viene chiamata la servlet `Logout`, che invalida la sessione (se non è già nulla) e reindirizza l'utente alla pagina di login.

- **Caricare una canzone**



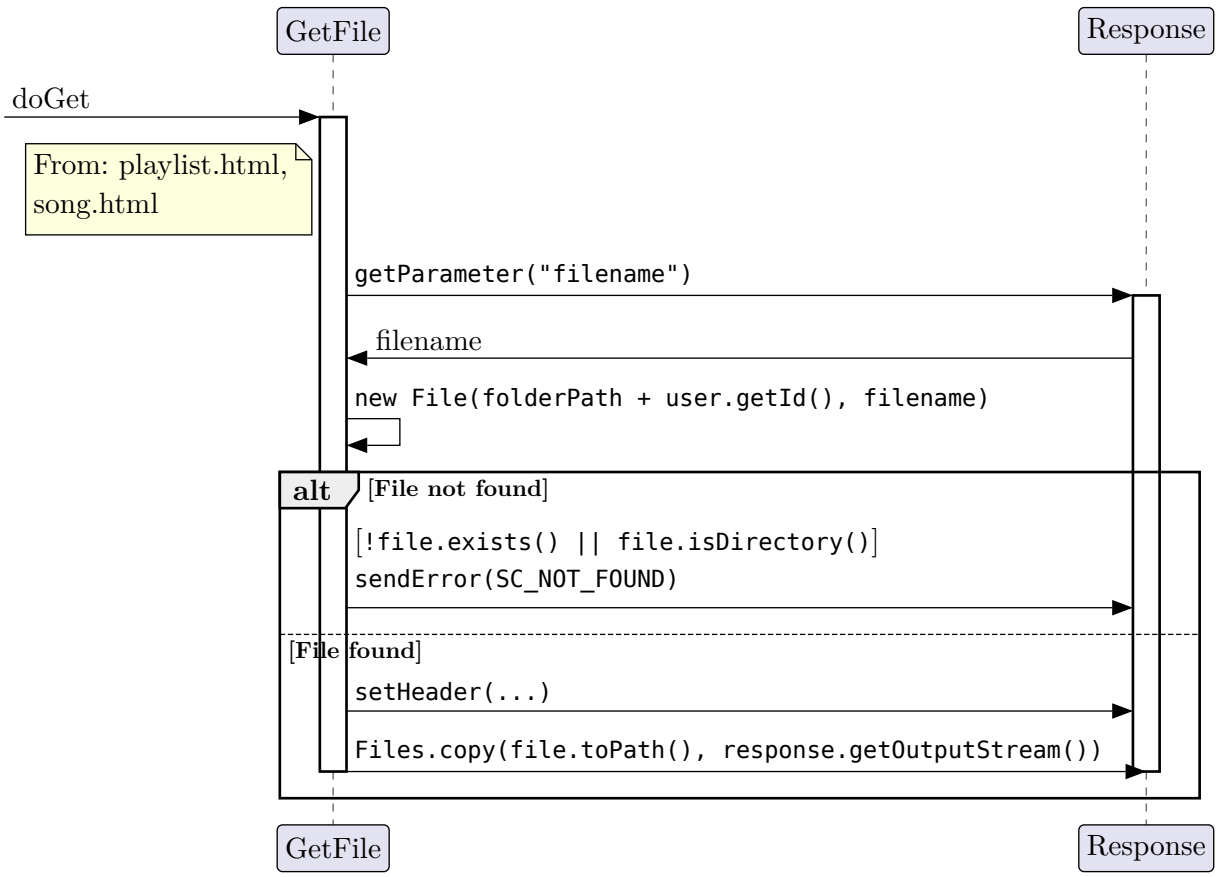
Quando l'utente invia il form per creare una canzone (tutti i campi sono `required`) viene controllato che i file d'immagine dell'album e musicale non sia nulli o vuoti e che siano del tipo giusto. Successivamente, se tutto il resto degli attributi è valido (non sono nulli e l'anno è positivo e non nel futuro) viene aggiornato il database e vengono copiati i file nell'apposita cartella dell'utente. Infine, l'utente viene reindirizzato alla homepage.

- **Creare una playlist**



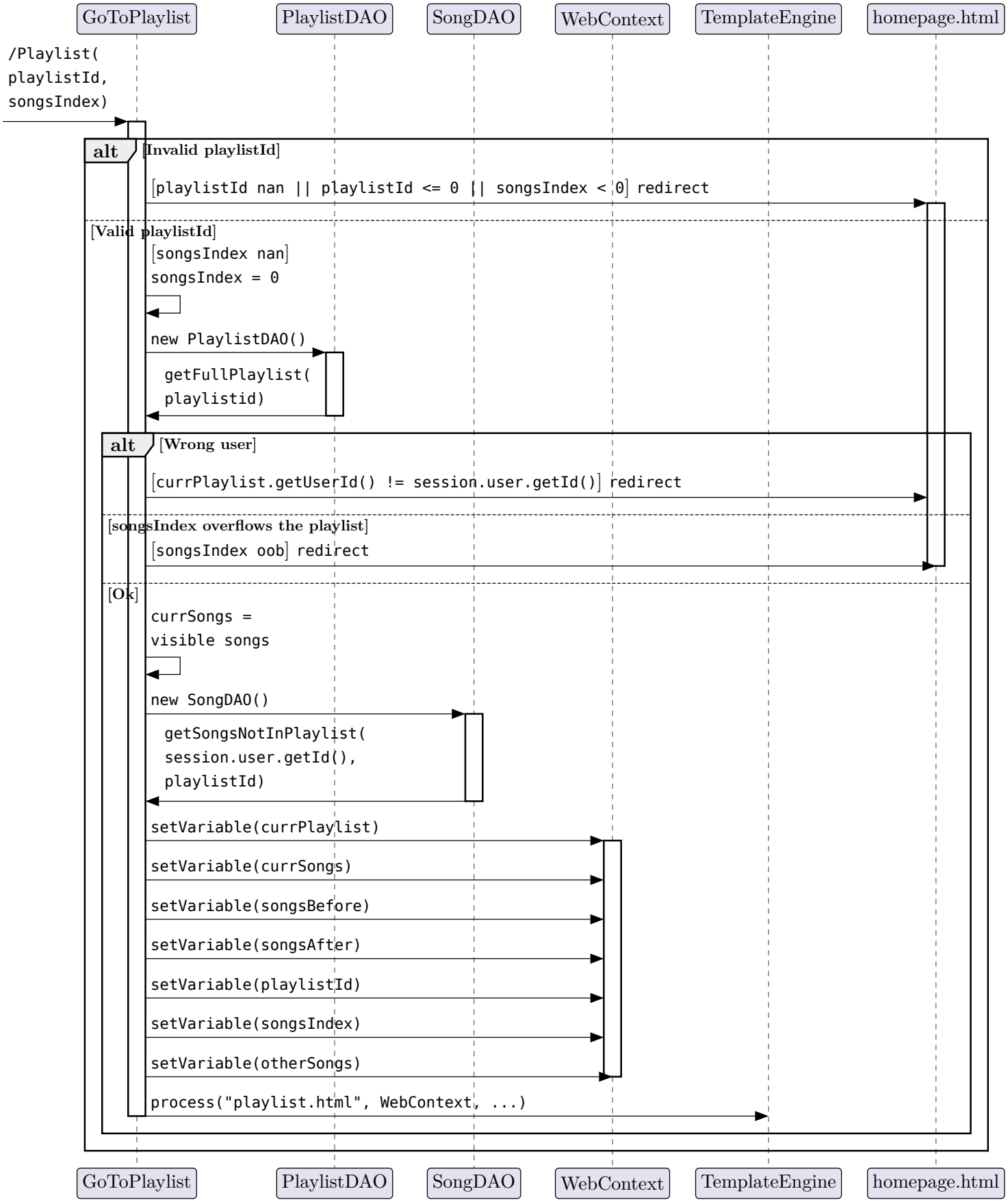
Quando l'utente invia il form per creare una nuova playlist (il campo del titolo è `required`) viene controllato che il titolo non sia nullo. Viene poi chiamato il `SongDAO` che restituisce una lista degli id delle canzoni dell'utente e controlla quali tra questi si trova nella richiesta: se viene trovato, viene aggiunto alla lista di canzoni da aggiungere alla playlist. Una volta terminato questo controllo, il `PlaylistDAO` crea la nuova playlist e l'utente è reindirizzato alla homepage.

Recuperare un file



Quando la pagina web deve mostrare un contenuto multimediale, chiama questa servlet, che cerca il file richiesto nella cartella dell'utente. Se non viene trovato, manda un errore, altrimenti renderizza il file tramite l'output stream.

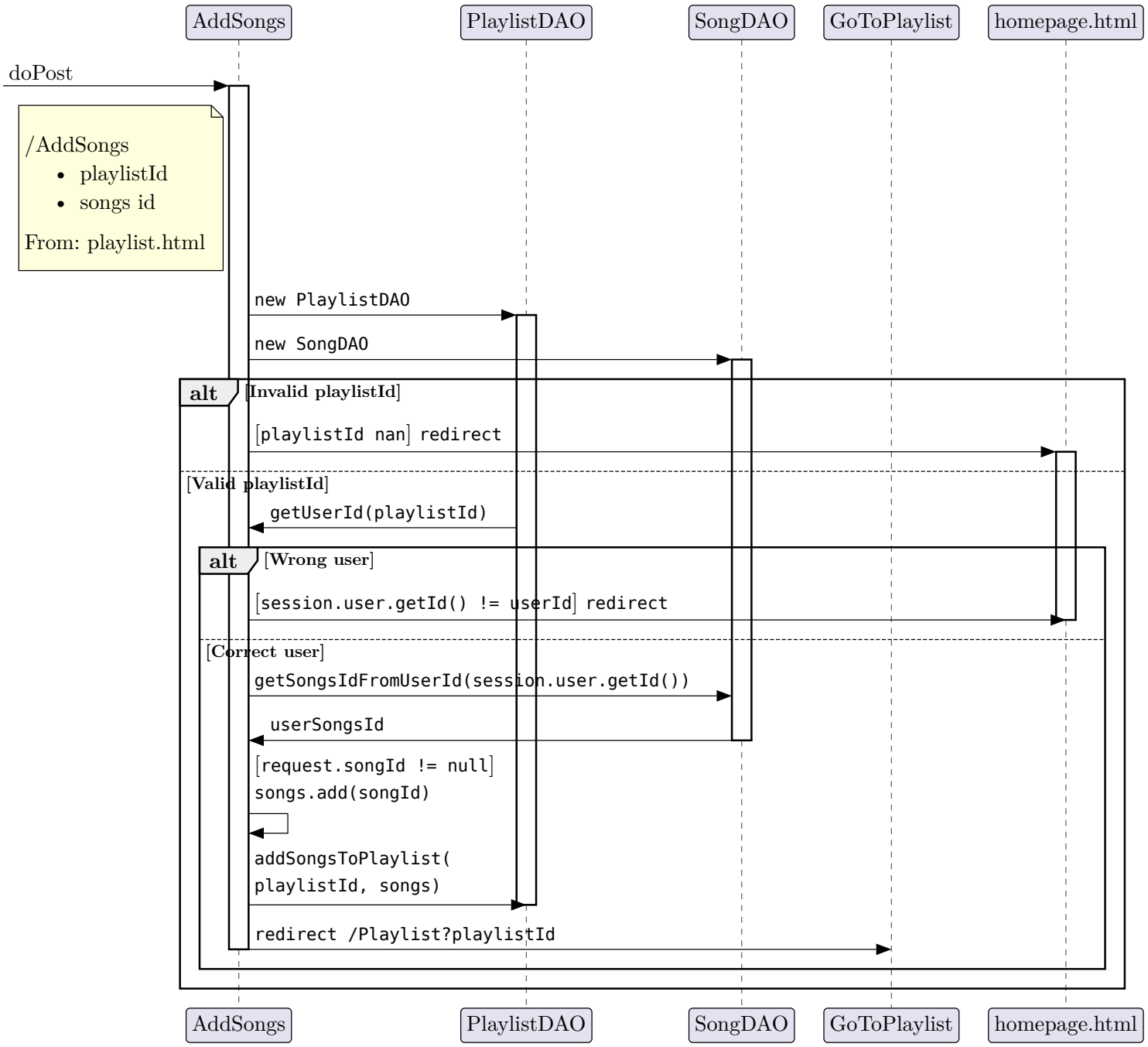
Andare alla pagina della playlist



Quando l'utente clicca sul nome di una playlist o clicca l'apposito pulsante nella pagina di una canzone, viene reindirizzato a questa pagina. Se l'id della playlist non è valido o è associato a una playlist non dell'utente corrente, viene reindirizzato alla homepage. Se l'indice non è un numero, gli viene dato il valore di default `0`, altrimenti se è non valido l'utente viene reindirizzato alla homepage. Se tutti i controlli hanno esito positivo, viene recuperata la playlist dalla `PlaylistDAO` e i brani dell'utente che non appartengono a quella playlist dal `SongDAO`. Dai brani della playlist vengono selezionati quelli «puntati» dall'indice (es. `songsIndex = 0` ⇒ `currSongs = [song0, song1, song2, song3, song4]`) e vengono assegnati ai valori booleani `songsBefore` e `songsAfter` valori appropriati in base all'indice. Tutte le variabili vengono poi caricate nel contesto e il template engine carica la pagina `playlist.html` con le informazioni necessarie:

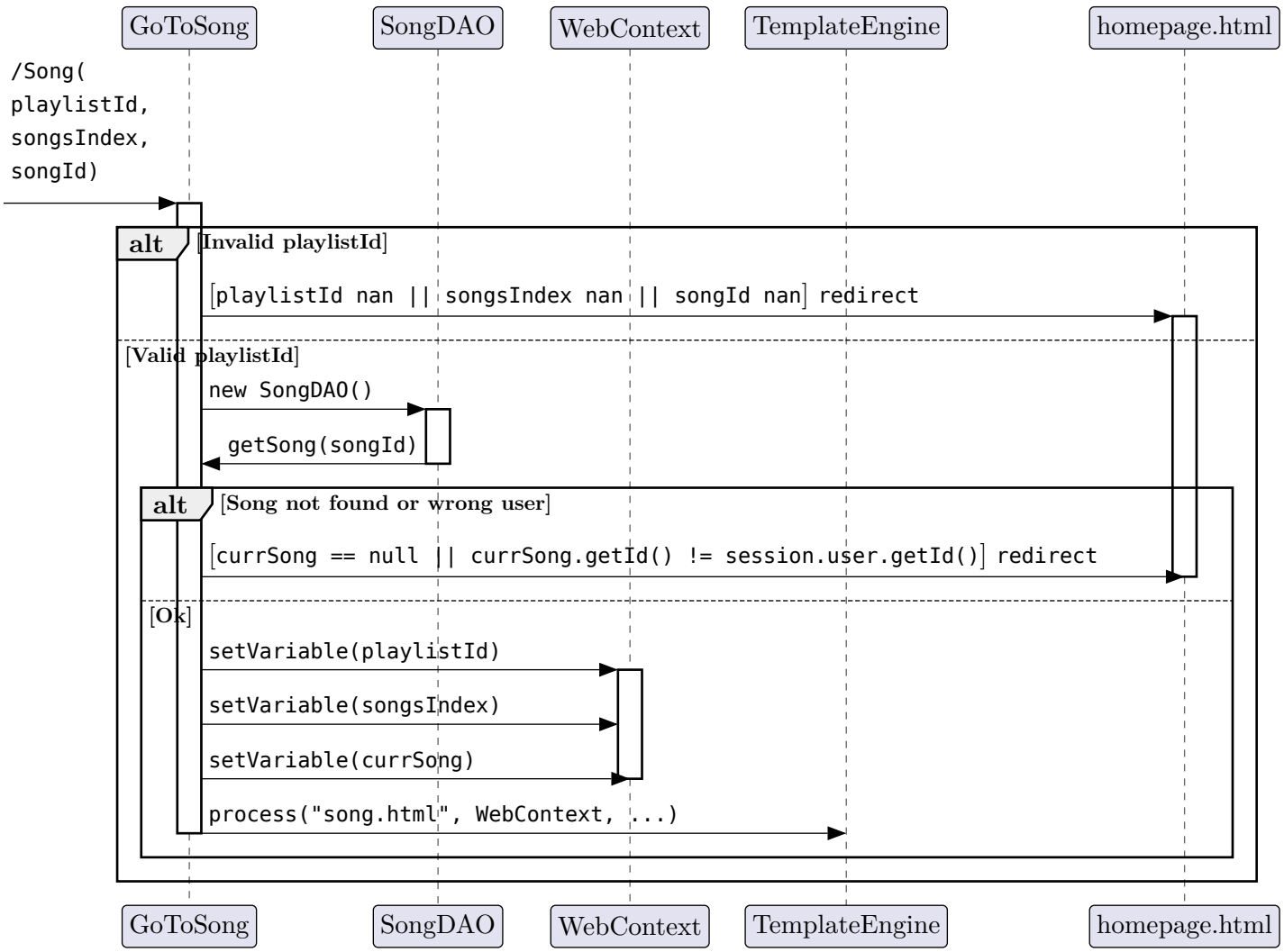
- Dalla playlist recupera il titolo e la data di creazione, che vengono mostrate a inizio pagina;
- Se la playlist è vuota, viene stampato un messaggio che consiglia di usare il form apposito per caricare brani alla playlist; altrimenti, vengono stampati i brani in base all'indice, disposti in una tabella di una riga e 5 colonne. Ogni cella ha il titolo del brano (che funge da link per la relativa pagina) e l'immagine dell'album (caricata tramite una chiamata alla `GetFile` servlet);
- I pulsanti per vedere i brani precedenti/successivi vengono mostrati in base alle variabili:
  - Se l'indice è 0, allora `songsBefore` è falso, e il pulsante non viene renderizzato, viceversa se l'indice è 0;
  - Se l'indice punta all'ultimo «gruppo di 5», allora `songsAfter` è falso, e il pulsante non viene renderizzato, viceversa se l'indice punta a un altro gruppo;
- Se la playlist contiene già tutte le canzoni dell'utente, viene stampato un messaggio che consiglia di caricare nuovi brani dalla homepage. Altrimenti, è presente un form da cui è possibile selezionare tutte le canzoni non già presenti nella playlist per aggiungerle.

Aggiungere canzoni alla playlist



Quando l'utente invia il form per aggiungere brani alla playlist, viene controllato che il playlist id della request sia valido: in caso negativo, l'utente viene reindirizzato alla homepage (idem se l'id è di una playlist non dell'attuale utente). Se l'id è valido si recupera la lista degli id dei brani dell'utente e viene controllata la request come descritto nel sequence diagram **Creare una playlist**: poi la playlist viene aggiornata tramite il `PlaylistDAO` e l'utente viene reindirizzato alla pagina della playlist.

Andare alla pagina della canzone



Quando l'utente clicca sul titolo di una canzone, viene reindirizzato a questa servlet. Se l'id della plalist, l'indice o l'id del brano non sono numeri, l'utente viene reindirizzato alla homepage. `SongDAO` viene chiamato per controllare che una canzone con quell'id esista e che appartenga allo user corrente: in caso negativo, viene reindirizzato alla homepage. Se tutti i controlli vanno bene, vengono salvati i dati nel contesto e caricata `song.html`:

- Tutte le informazioni della canzone vengono mostrate, insieme alla foto associata e al suo lettore musicale (anche questi file vengono recuperati tramite la servlet `GetFile`);
- L'id della playlist e l'indice servono quando l'utente usa il pulsante per tornare alla playlist: così facendo, ritorna alla pagina dalla quale ha cliccato la canzone.

3. Documentazione ver. Javascript

3.a. Analisi requisiti dati

L'applicazione deve consentire all'utente di **riordinare le playlist** con un criterio personalizzato diverso da quello di default. Dalla HOME con un link associato a ogni playlist si accede a una finestra modale RIORDINO, che mostra la lista completa dei brani della playlist ordinati secondo il criterio corrente (personalizzato o di default). L'utente può trascinare il titolo di un brano nell'elenco e di collocarlo in una **posizione** diversa per realizzare l'ordinamento che desidera, senza invocare il server. Quando l'utente ha raggiunto l'ordinamento desiderato, usa un bottone «salva ordinamento», per memorizzare la sequenza sul server. Ai successivi accessi, l'ordinamento personalizzato è usato al posto di quello di default. Un brano aggiunto a una playlist con ordinamento personalizzato è inserito nell'ultima posizione.

Legenda:

- **Entità**
- **Attributi**
- **Relazioni**.

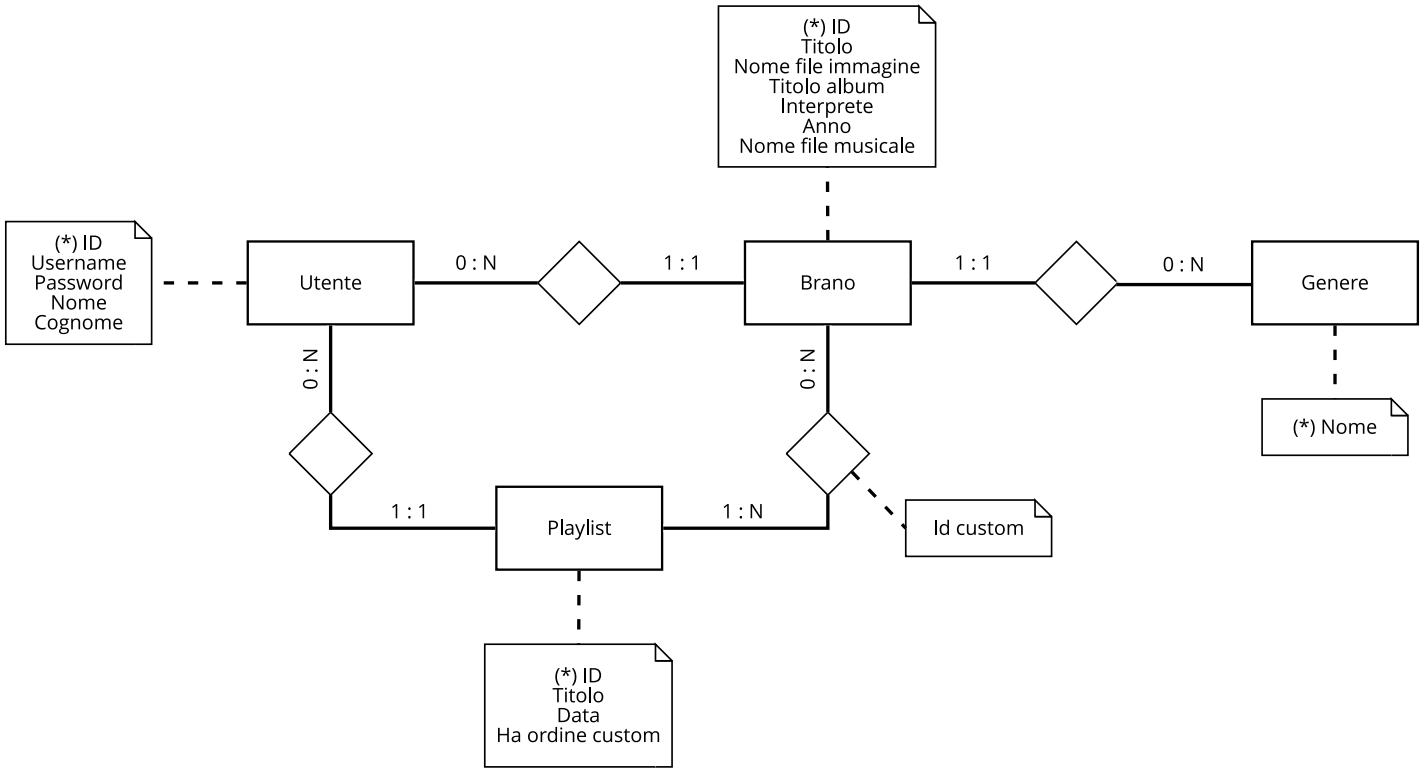


Diagramma entità-relazioni

```sql
create table users
(
    id       int auto_increment,
    username varchar(32) not null unique,
    password varchar(32) not null,
    name     varchar(32) not null,
    surname  varchar(32) not null,
    primary key (id)
);

create table genres
(
    name varchar(32),
    primary key (name)
);

create table songs
(
    id              int auto_increment,
    user_id         int          not null,
    title           varchar(256) not null,
    image_file_name varchar(256) not null,
    album_title     varchar(256) not null,
    performer       varchar(256) not null,
    year            int          not null check ( year > 0 ),
    genre           varchar(256) not null,
    music_file_name varchar(256) not null,
    primary key (id),
    foreign key (user_id) references users (id) on update cascade on delete no action,
    foreign key (genre) references genres (name) on update cascade on delete no action,
    unique (user_id, music_file_name),
    unique (user_id, title)
);

create table playlists
(
    id               int auto_increment,
    user_id          int          not null,
    title            varchar(256) not null,
    date             date         not null default current_date,
    has_custom_order boolean      not null default false,
    primary key (id),
    unique (user_id, title)
);

create table playlist_contents
(
    playlist  int,
    song      int,
    custom_id int default null,
    primary key (playlist, song),
    foreign key (playlist) references playlists (id) on update cascade on delete no
action,
    foreign key (song) references songs (id) on update cascade on delete no action
);
```

Database design

3.b. Analisi requisiti d'applicazione

Si realizzi un'applicazione client server web che modifica le specifiche precedenti come segue:

- Dopo il **login dell'utente**, l'intera applicazione è realizzata con un'**unica pagina**
- Ogni interazione dell'utente è gestita senza ricaricare completamente la pagina, ma produce l'invocazione asincrona del server e l'eventuale modifica del contenuto da aggiornare a seguito dell'evento;
- L'evento di visualizzazione del blocco precedente/successivo è gestito a lato client senza generare una richiesta al server;
- L'applicazione deve consentire all'utente di riordinare le playlist con un criterio personalizzato diverso da quello di default. Dalla HOME con un link associato a ogni playlist si accede a una finestra modale **RIORDINO**, che mostra **la lista completa dei brani della playlist ordinati secondo il criterio corrente** (personalizzato o di default). L'utente può **trascinare il titolo di un brano nell'elenco e di collocarlo in una posizione diversa** per realizzare l'ordinamento che desidera, senza invocare il server. Quando l'utente ha raggiunto l'ordinamento desiderato, usa un **bottone «salva ordinamento»**, per **memorizzare la sequenza sul server**. Ai successivi accessi, **l'ordinamento personalizzato è usato al posto di quello di default**. **Un brano aggiunto a una playlist con ordinamento personalizzato** è **inserito nell'ultima posizione**.

Legenda:

- **Pagine**;
- **Componenti**;
- **Eventi**;
- **Azioni**.

3.c. Aggiunta alle specifiche

- Funzione di logout, accessibile tramite un pulsante dalle pagine di home, playlist e canzone;
- Funzione per tornare alla homepage, accessibile tramite un pulsante dalle pagine di playlist e canzone;
- Funzione per tornare alla pagina della playlist originale, accessibile tramite un pulsante dalla pagina della canzone;
- Messaggio di «benvenuto» quando l'utente è nella home page;
- Nella pagina della playlist viene mostrata la data in cui è stata creata;
- Possibilità di creare una playlist senza brani;
- Istruzioni sul creare una playlist nella homepage se l'utente non ha playlist associate;
- Istruzioni sull'aggiungere brani alla playlist se vuota nella pagina della playlist;
- Istruzioni sul caricare canzoni nella pagina della playlist se l'utente o ha aggiunto tutti i suoi brani alla playlist o non ha brani associati;
- Possibilità di annullare il riordino di una playlist cliccando l'apposito pulsante o al di fuori del modal.

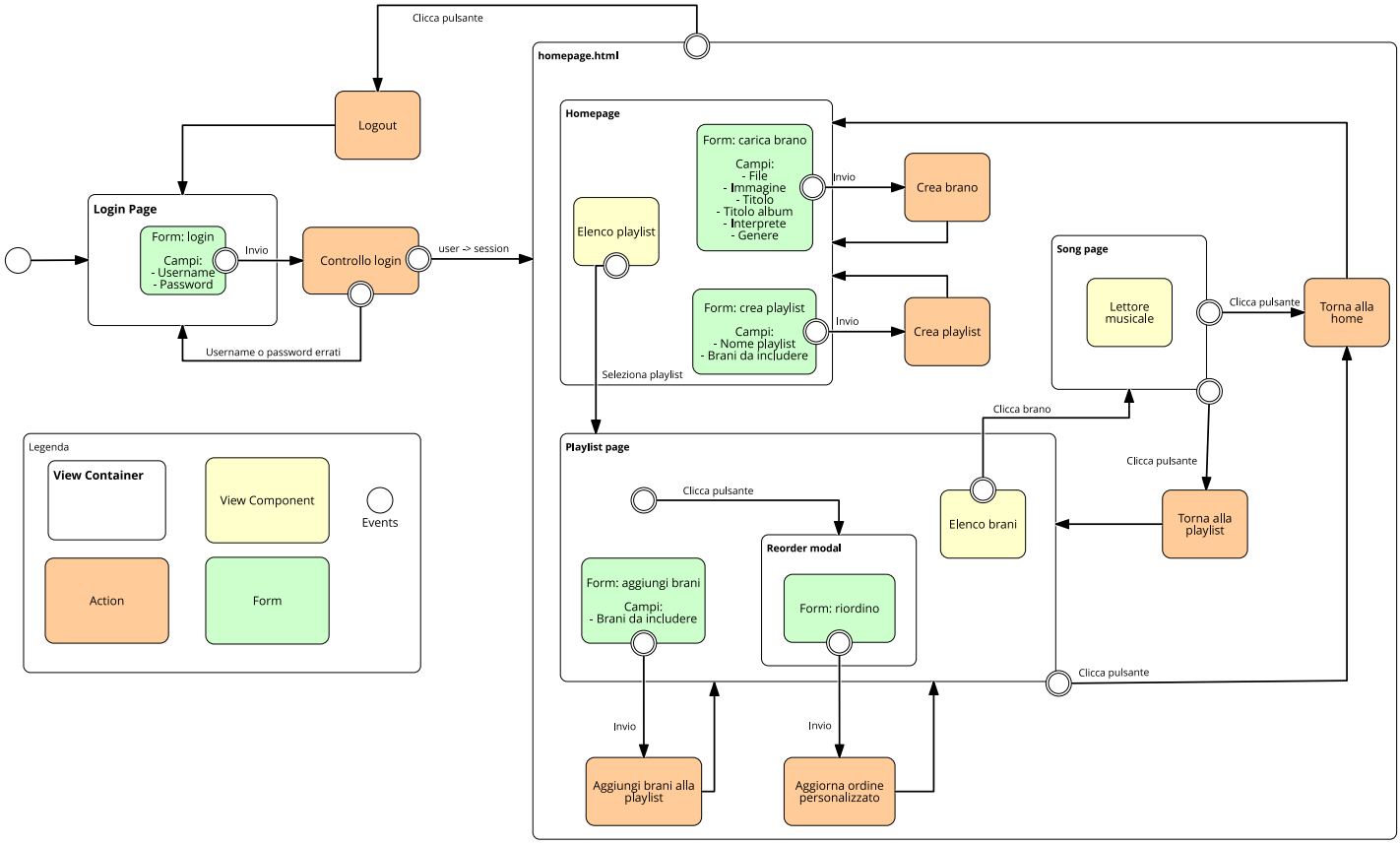
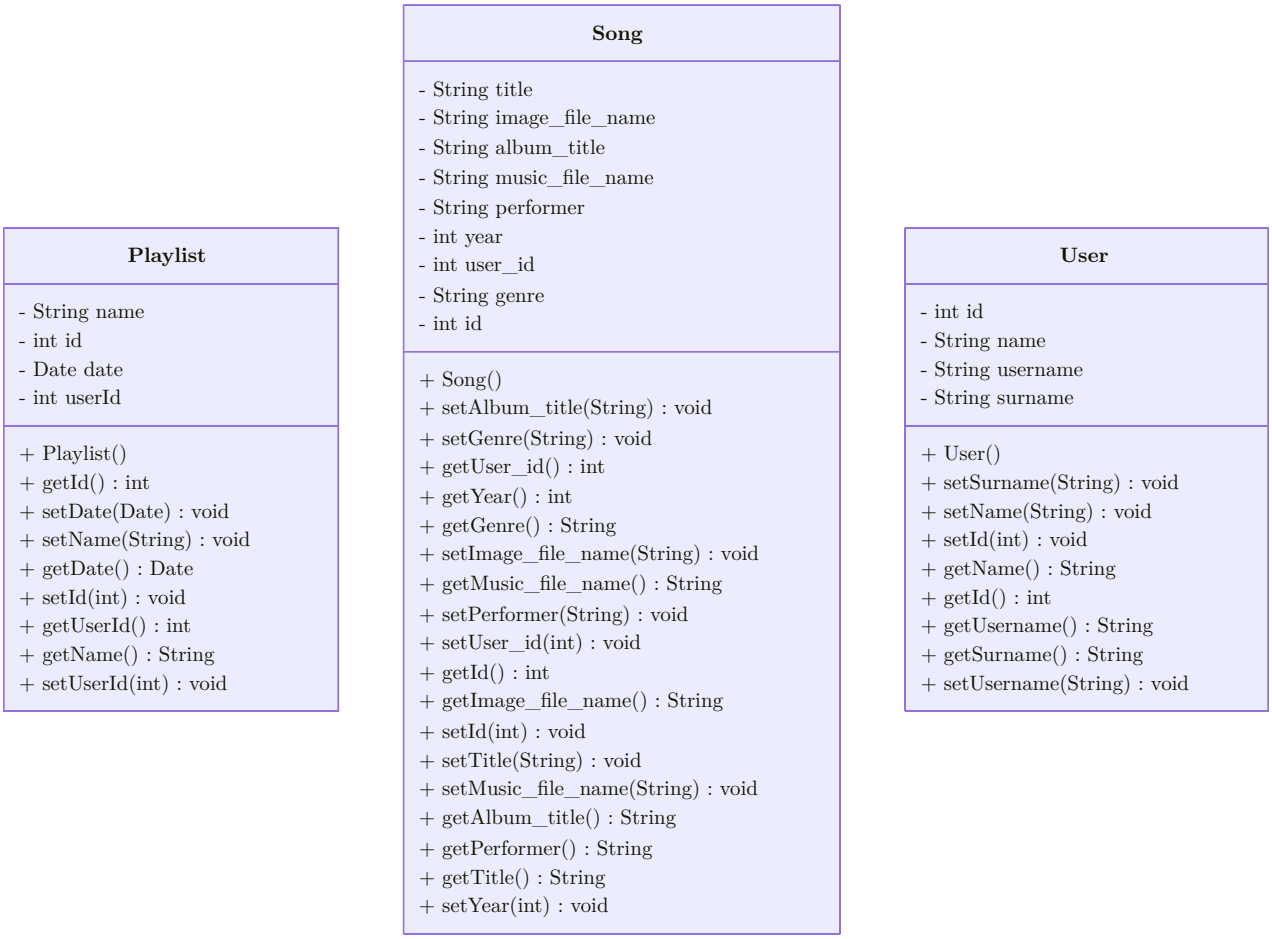


Diagramma IFML

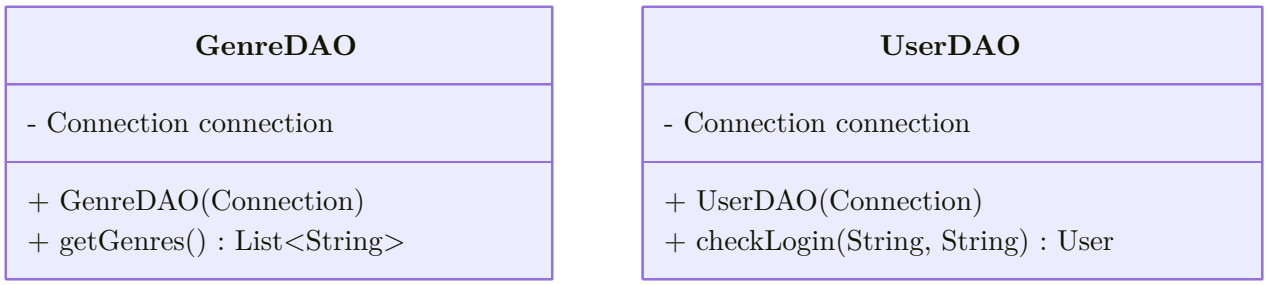
3.d. Componenti e viste

3.d.a. Beans



I beans sono uguali alla **versione precedente**, tranne la playlist che non ha l'attributo `Songs`: questo perché il recupero delle canzoni della playlist è stato spostato in `SongDAO`, che tiene conto della presenza o meno dell'ordine personalizzato.

3.d.b. DAOs



Questi due DAO sono uguali alla **versione precedente**.

PlaylistDAO
- Connection connection
+ PlaylistDAO(Connection) + insertPlaylist(int, String, List<Integer>) : void + addSongsToPlaylist(int, List<Integer>) : void + getPlaylist(int) : Playlist + getPlaylists(int) : List<Playlist> + getUserId(int) : int + updateCustomOrder(int, List<Integer>) : void + hasCustomOrder(int) : boolean + getPlaylistId(int, String) : int

- `getPlaylist(int playlistId)`: l'equivalente di `getFullPlaylist` della **versione precedente**, con la differenza che non recupera direttamente le canzoni della playlist;
- `updateCustomOrder(int playlistId, List<Integer> songsId)`: data una lista di id di brani e l'id della playlist, aggiorna il suo ordine personalizzato in base all'ordine degli id nella lista (il primo id sarà il primo brano visualizzato, etc…);
- `hasCustomOrder(int playlistId)`: ritorna vero se la playlist ha un ordine personalizzato, falso altrimenti.

Il resto dei metodi sono uguali alla **versione precedente**.

SongDAO
- Connection connection
+ SongDAO(Connection) + getSongsIdFromUserId(int) : List<Integer> + getAllSongsFromPlaylist(int) : List<Song> + getAllSongsFromUserId(int) : List<Song> + insertSong(int, String, String, String, String, int, String, String) : void + getSongsIdFromPlaylist(int) : List<Integer> - getSongListFromResultSet(ResultSet) : List<Song> + getSongsNotInPlaylist(int, int) : List<Song> + getSong(int) : Song - getSongFromResultSet(ResultSet) : Song

Tutti i metodi sono simili alla **versione html**, tranne `getAllSongsFromPlaylist`, che controlla se la playlist ha un ordine personalizzato per decidere quale tipo di ordinamento usare.

3.d.c. Controllers

- `AddSongsToPlaylist`;
- `CheckLogin`;
- `CreatePlaylist`;
- `GetGenres`;
- `GetPlaylist`;
- `GetPlaylists`;
- `GetSong`;
- `GetSongsByUserID`;
- `GetSongsFromPlaylist`;
- `GetSongsNotInPlaylist`;
- `GetUserData`;
- `Logout`;
- `UpdateCustomOrder`;
- `UploadSong`.

3.d.d. Filters

- `LoginChecker`: quando l'utente tenta di accedere a `homepage.html`, il filtro controlla che la sessione non sia «nuova», e richiede l'attributo `user`. Se la sessione è nuova o l'`user` è `null`, l'utente viene indirizzato alla pagina di login, altrimenti la servlet procede nel suo compito.

3.d.e. Utils

- `getConnection(ServletContext context)`: **come sopra**
- `getFileEncoding(String relativeFilePath, ServletContext context)`: metodo che prende il file dal local storage e lo encripta in una stringa «base64». Dal contesto prende il percorso che punta alla cartella con tutte le risorse degli utenti, e il percorso relativo viene aggiunto per puntare al file specifico.

3.d.f. Html files

- `login.html` (welcome-file);
- `homepage.html`.

3.d.g. JS

- `filter`: controlla che l'utente abbia fatto il login;
- `homepageButton`: quando l'utente clicca l'`homepage_button`, chiama la funzione `showHomepage()`, dato che non è necessario inizializzarla di nuovo;
- `homepageManager`: si occupa di inizializzare la homepage (`homepageInit()`), aggiornare la lista di playlist dell'utente (`updatePlaylists()`), aggiornare il form per creare una nuova playlist con le canzoni dell'utente (`updateCreatePlaylistForm()`), renderizzare la homepage (`showHomepage()`) e controllare la correttezza dei form prima di chiamare la servlet;
- `loginManagement`: si occupa di gestire il login dell'utente, controllando che il form sia valido prima di chiamare la servlet;
- `logout`: quando l'utente clicca il bottone `logout`, fa una chiamata alla relativa servlet e pulisce la session storage;
- `modalManager`: si occupa di gestire le funzioni del modal, come renderizzarlo, chiuderlo (`closeModal()`), riordinare la playlist con il drag & drop e confermare il nuovo ordine;
- `playlistPageManager`: si occupa di inizializzare la pagina della playlist (`playlistPageInit()`), renderizzarla (`showPlaylistPage()`), cambiare le canzoni visualizzate tramite i pulsanti «precedenti» e «successive» (`showVisibleSongs()`) e gestire il form `add_songs_to_playlist`;
- `songPageManager`: si occupa di renderizzare la pagina del brano e il pulsante per tornare alla playlist;
- `utils.js`: si occupa di gestire le chiamate con le servlet tramite la funzione `makeCall()`.

3.e. Eventi e azioni

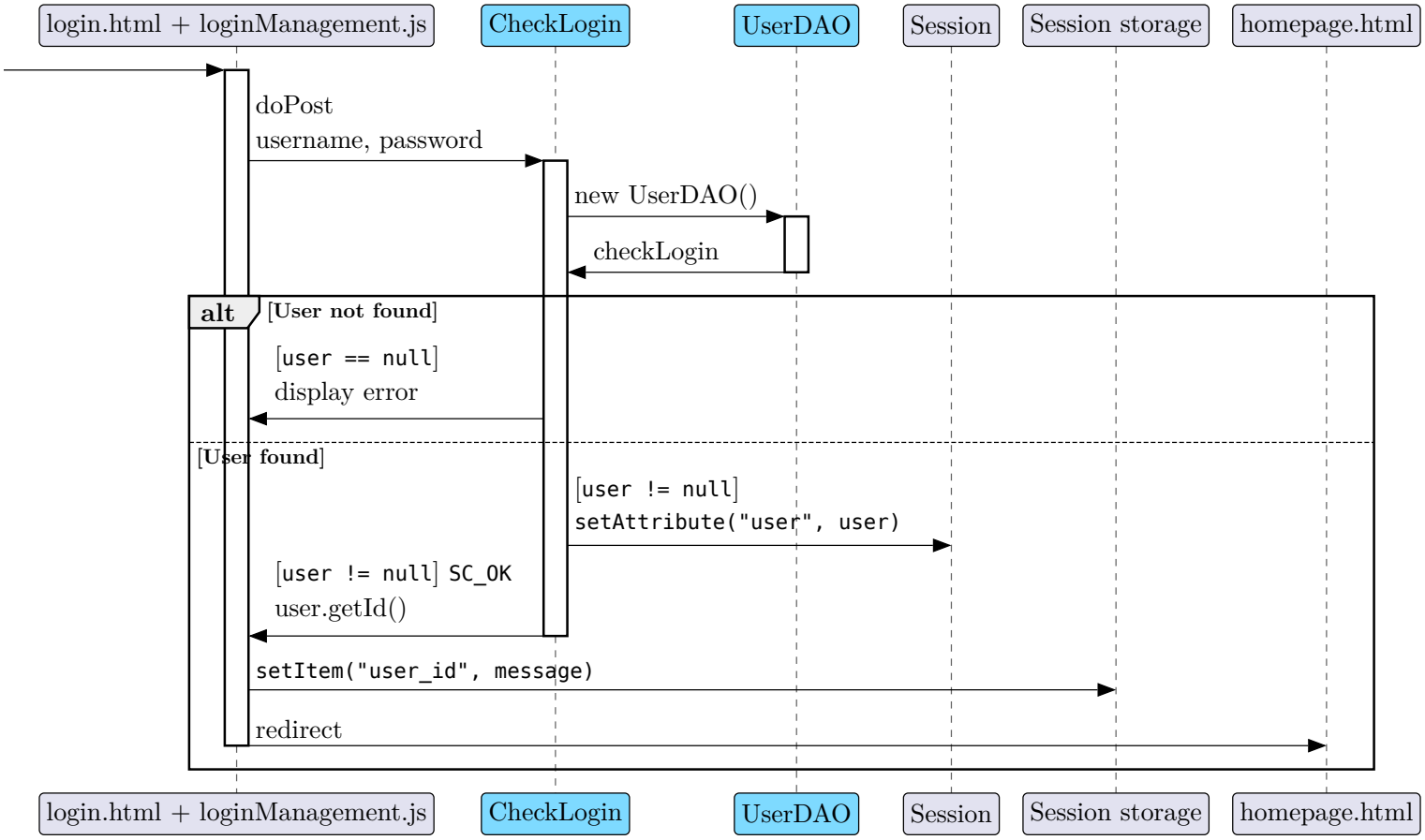
Client side		Server side	
Evento	**Azione**	**Evento**	**Azione**
Index ⇒ login form ⇒ submit	Controllo dati	POST: username, password	Controllo credenziali
Homepage ⇒ primo caricamento	Aggiorna view	GET	Recupera informazioni utente e generi
Homepage ⇒ primo caricamento o creazione playlist	Aggiorna elenco playlist	GET	Recupera playlist dell'utente
Homepage ⇒ primo caricamento o aggiunta brano	Aggiorna create playlist form	GET	Recupera canzoni dell'utente
Homepage ⇒ upload song form ⇒ submit	Controllo dati	POST: titolo, album, interprete, anno, genere, file musicale, immagine	Aggiunta brano
Homepage ⇒ create playlist form ⇒ submit	Controllo dati	POST: titolo, elenco brani da aggiungere	Creazione playlist
Homepage ⇒ elenco playlist ⇒ seleziona playlist	Aggiorna view e visualizza playlist page	GET: playlistId	Recupero canzoni presenti nella playlist e non
Playlist page ⇒ add songs form ⇒ submit	Controllo dati	POST: elenco brani da aggiungere	Aggiunta brani alla playlist
Playlist page ⇒ riordino	Aggiorna view e mostra modal di riordino	GET: playlistId	Recupero brani presenti nella playlist
Modal riordino ⇒ riordino ⇒ submit	Controllo dati	POST: brani ordinati	Salva l'ordine personalizzato
Modal riordino ⇒ annulla riordino/click fuori dal modal	Nascondi modal di riordino	-	-
Playlist page ⇒ successive/precedenti canzoni ⇒ submit	Aggiorna view	-	-
Playlist page ⇒ elenco brani ⇒ seleziona brano	Aggiorna view e visualizza song page	GET: songId	Recupero informazioni brano
Playlist/song page ⇒ homepage ⇒ submit	Aggiorna view e visualizza homepage	-	-
Song page ⇒ torna alla playlist ⇒ submit	Aggiorna view e visualizza playlist page	-	-
Logout	-	GET	Terminazione sessione

3.f. Controller ed Event handler

Client side		Server side	
Evento	**Controller**	**Evento**	**Controller**
Index ⇒ login form ⇒ submit	Funzione `makeCall`	POST: username, password	Servlet `CheckLogin`
Homepage ⇒ primo caricamento	Funzione `homepageInit`	GET	Servlet `GetUser` e `GetGenres`
Homepage ⇒ primo caricamento o creazione playlist	Funzione `updatePlaylists`	GET	Servlet `GetPlaylists`
Homepage ⇒ primo caricamento o aggiunta brano	Funzione `updCreatePlaylistForm`	GET	Servlet `GetSongsByUserID`
Homepage ⇒ upload song form ⇒ submit	Funzione `makeCall`	POST: titolo, album, interprete, anno, genere, file musicale, immagine	Servlet `UploadSong`
Homepage ⇒ create playlist form ⇒ submit	Funzione `makeCall`	POST: titolo, elenco brani da aggiungere	Servlet `CreatePlaylist`
Homepage ⇒ elenco playlist ⇒ seleziona playlist	Funzione `playlistPageInit`	GET: playlistId	Servlet `GetPlaylist`, `GetSongsFromPlaylist` e `GetSongsNotInPlaylist`
Playlist page ⇒ add songs form ⇒ submit	Funzione `makeCall`	POST: elenco brani da aggiungere	Servlet `AddSongsToPlaylist`
Playlist page ⇒ riordino	Funzione `showReorderPage`	GET: playlistId	Servlet `GetSongsFromPlaylist`
Modal riordino ⇒ riordino ⇒ submit	Funzione `makeCall`	POST: brani ordinati	Servlet `UpdateCustomOrder`
Modal riordino ⇒ annulla riordino/click fuori dal modal	Funzione `closeModal`	-	-
Playlist page ⇒ successive/precedenti canzoni ⇒ submit	Funzione `showVisibleSongs`	-	-
Playlist page ⇒ elenco brani ⇒ seleziona brano	Funzione `showSongPage`	GET: songId	Servlet `GetSong`
Playlist/song page ⇒ homepage ⇒ submit	Funzione `showHomepage`	-	-
Song page ⇒ torna alla playlist ⇒ submit	Funzione `showPlaylistPage`	-	-
Logout	-	GET	Servlet `Logout`

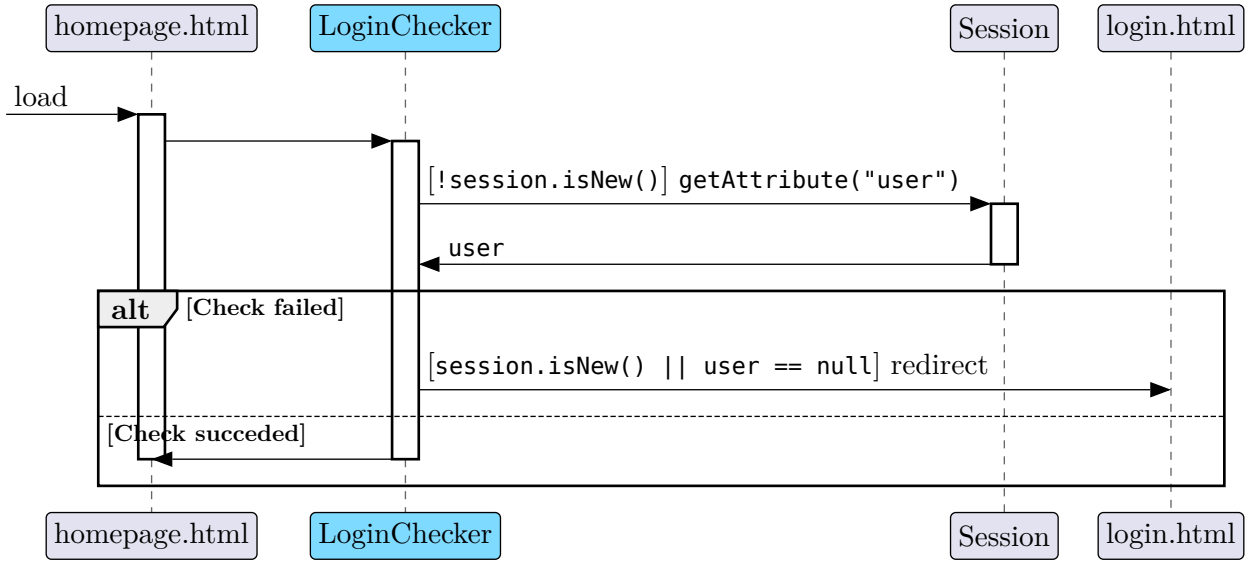
3.g. Sequence diagrams

Login



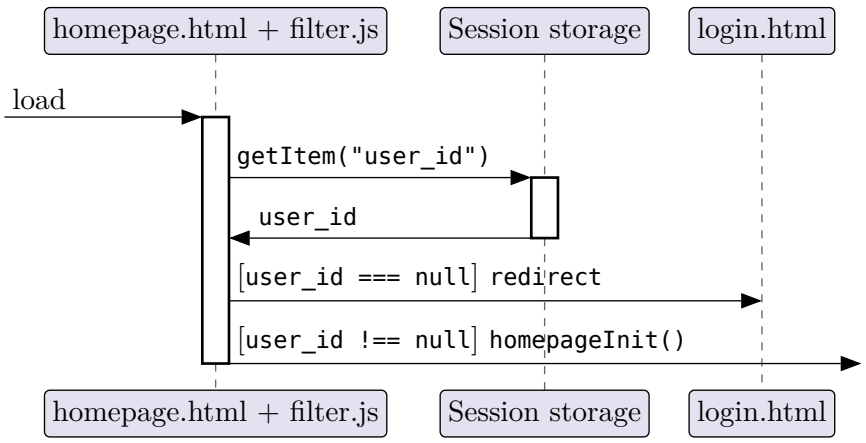
Simile alla [versione precedente]. A lato client, `loginManagement.js` controlla che il form sia valido prima di mandare la richiesta: se la servlet ritorna esito positivo, salva lo `user_id` nel session storage e reindirizza lo user alla homepage; se la servlet ritorna esito negativo, stampa un messaggio di errore.

Filtro utente



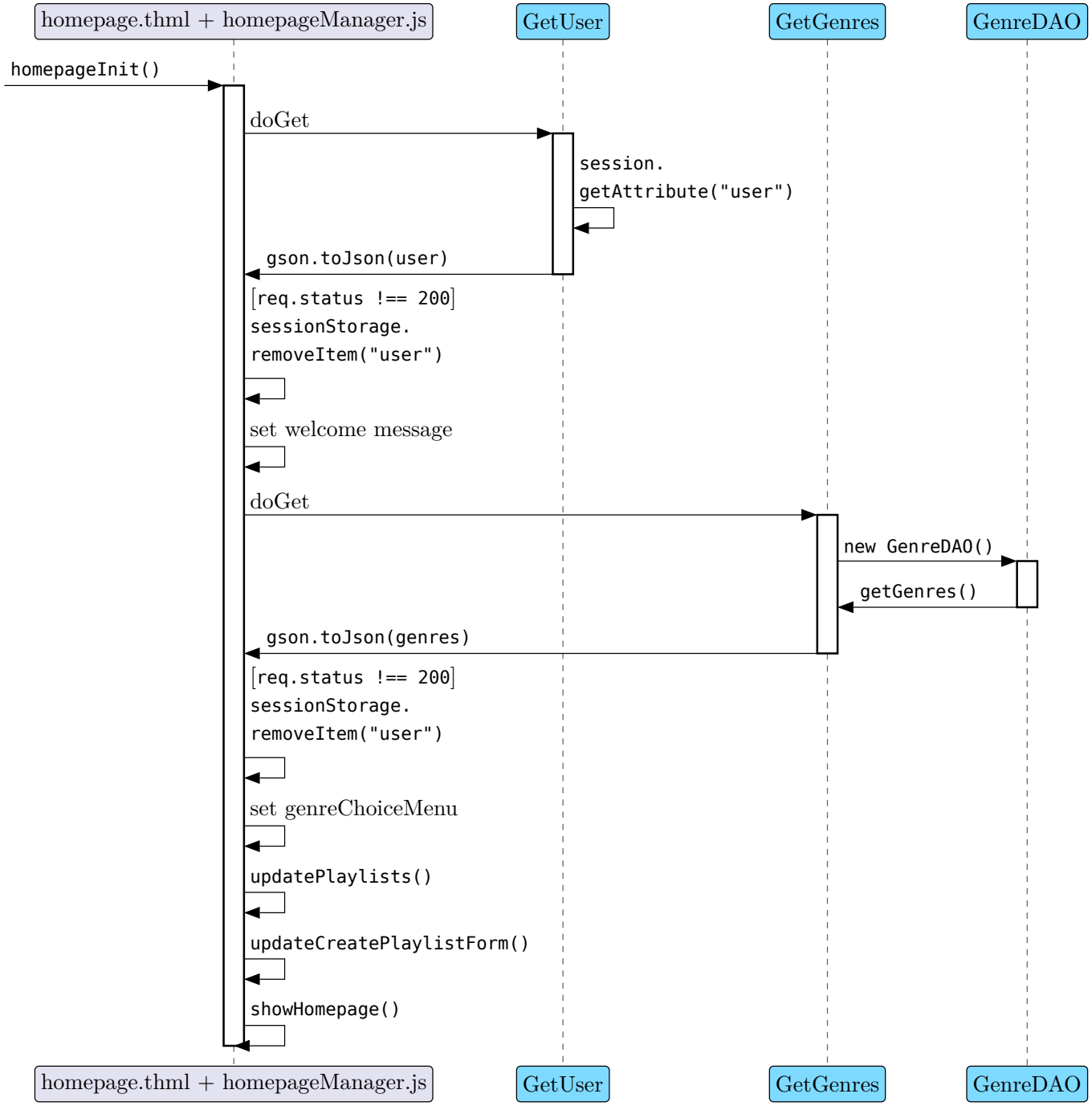
Al caricamento della homepage, il `LoginChecker` controlla che la sessione non sia nuova e che l'utente sia valido: se questi controlli hanno esito positivo, procede col caricamento, altrimenti reindirizza l'utente alla pagina di login.

Caricare la homepage



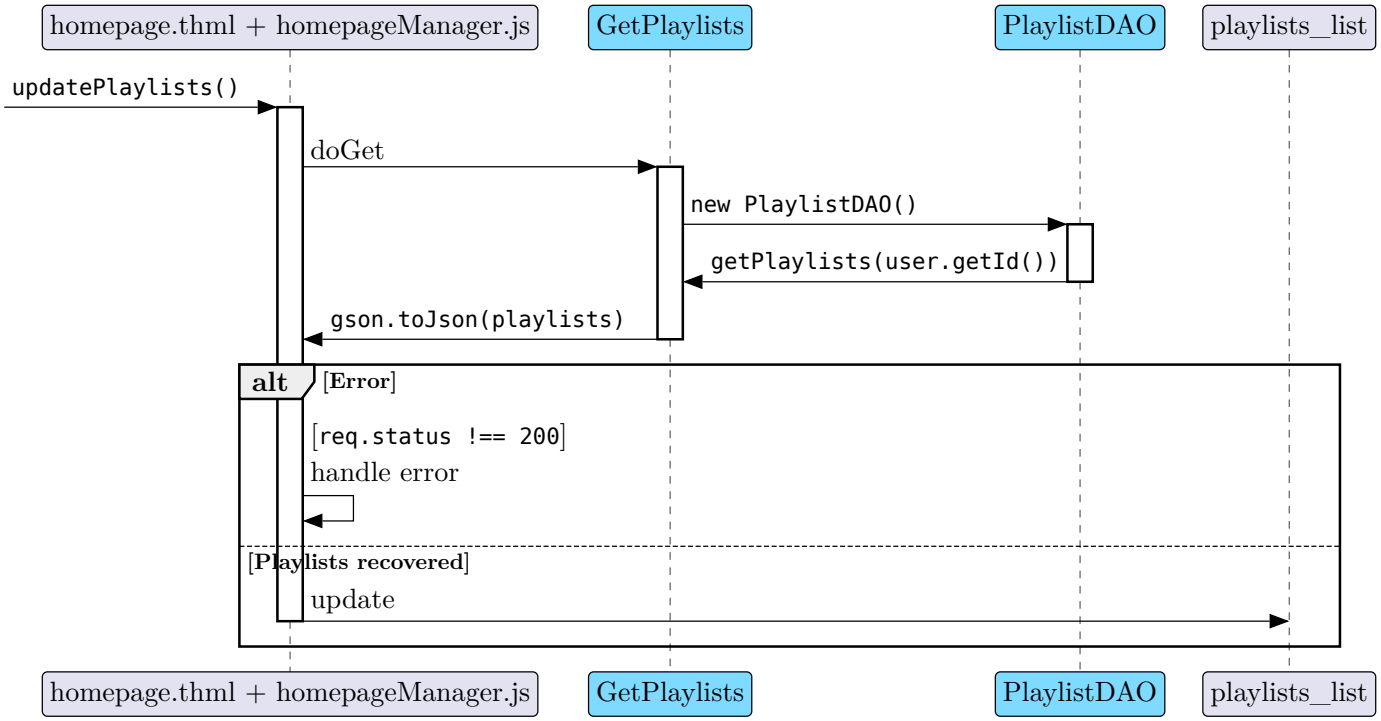
Al caricamento di `homepage.html`, controlla che l'attributo `user_id` non sia nullo: se lo è, reindirizza l'utente alla pagina di login, altrimenti renderizza la homepage chiamando `homepageInit`.

Inizializzare la homepage



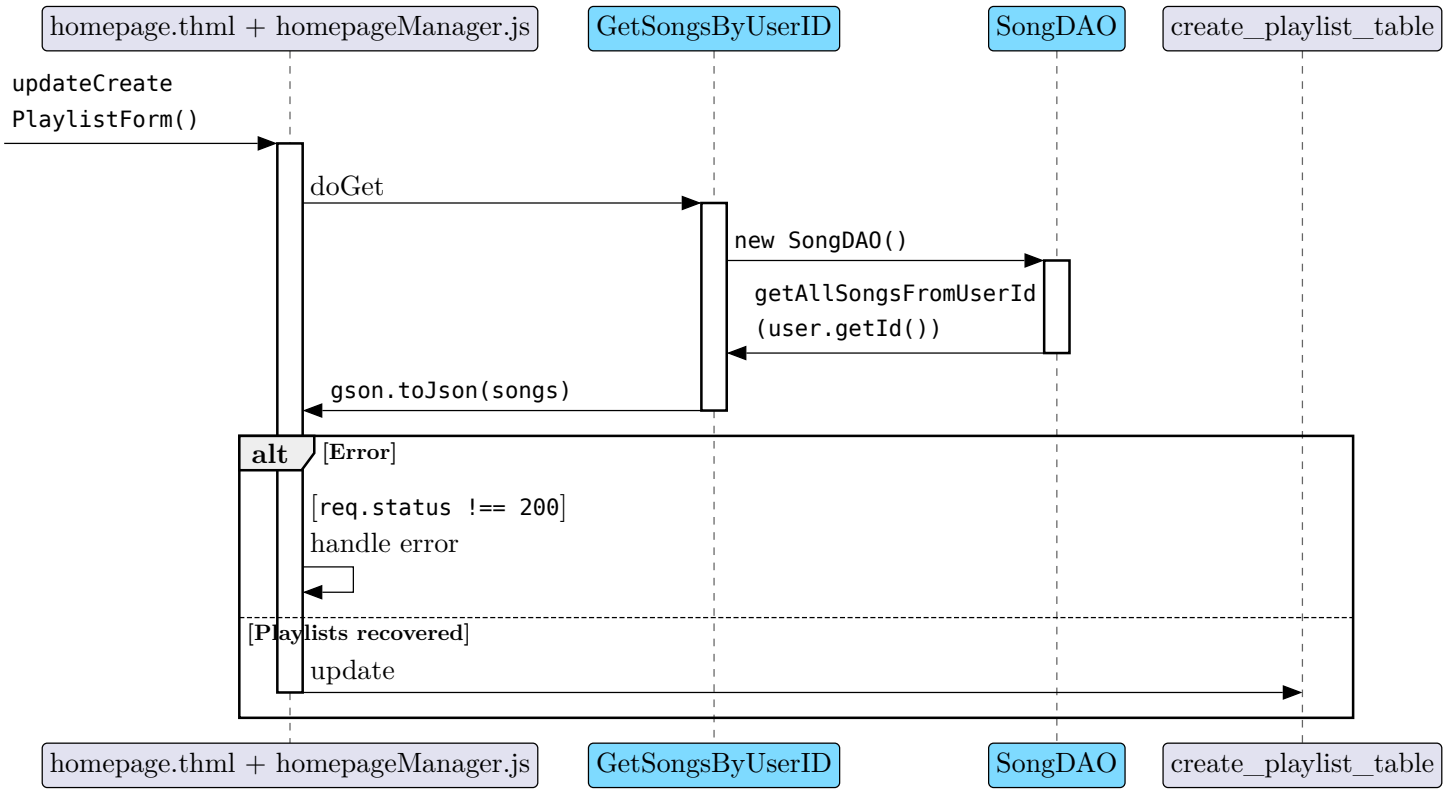
Quando la homepage viene caricata per la prima volta e passa il filtro, vengono chiamate le servlet `GetUser` e `GetGenres` per inizializzare il «messaggio di benvenuto» e il menu a tendina dei generi. Se queste chiamate hanno esito positivo, vengono poi chiamate le funzioni `updatePlaylists()` e `updateCreatePlaylistForm()` (separate da `homepageInit()` per permettere di chiamare le relative servlet solo quando è necessario); infine, chiama `showHomepage()` per renderizzare la homepage.

Aggiornare la lista di playlist



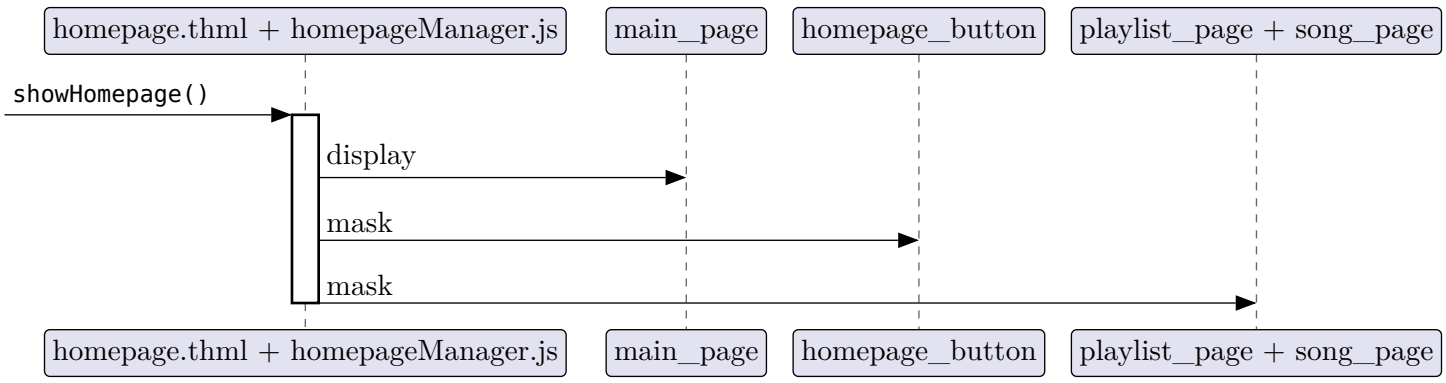
Quando è necessario aggiornare l'elenco di playlist, chiama la servlet `GetPlaylists` e ritorna le playlist come JSON. Se la risposta arriva correttamente, viene aggiornato l'elenco.

Aggiornare il form per creare una playlist

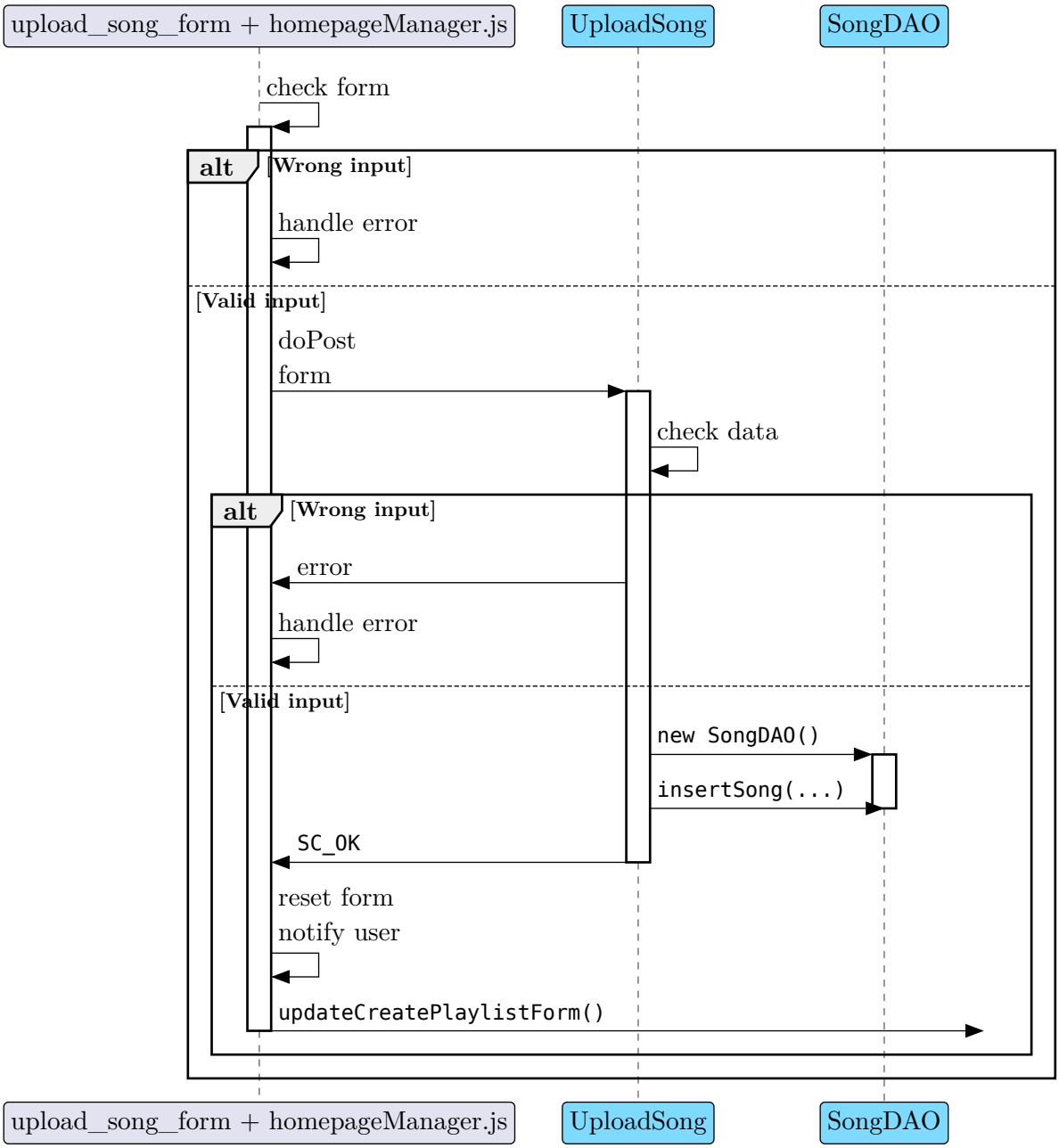


Quando è necessario aggiornare il form per creare playlist, chiama la servlet `GetSongsByUserID` e ritorna le canzoni come JSON. Se la risposta arriva correttamente, viene aggiornato il form.

Andare alla homepage



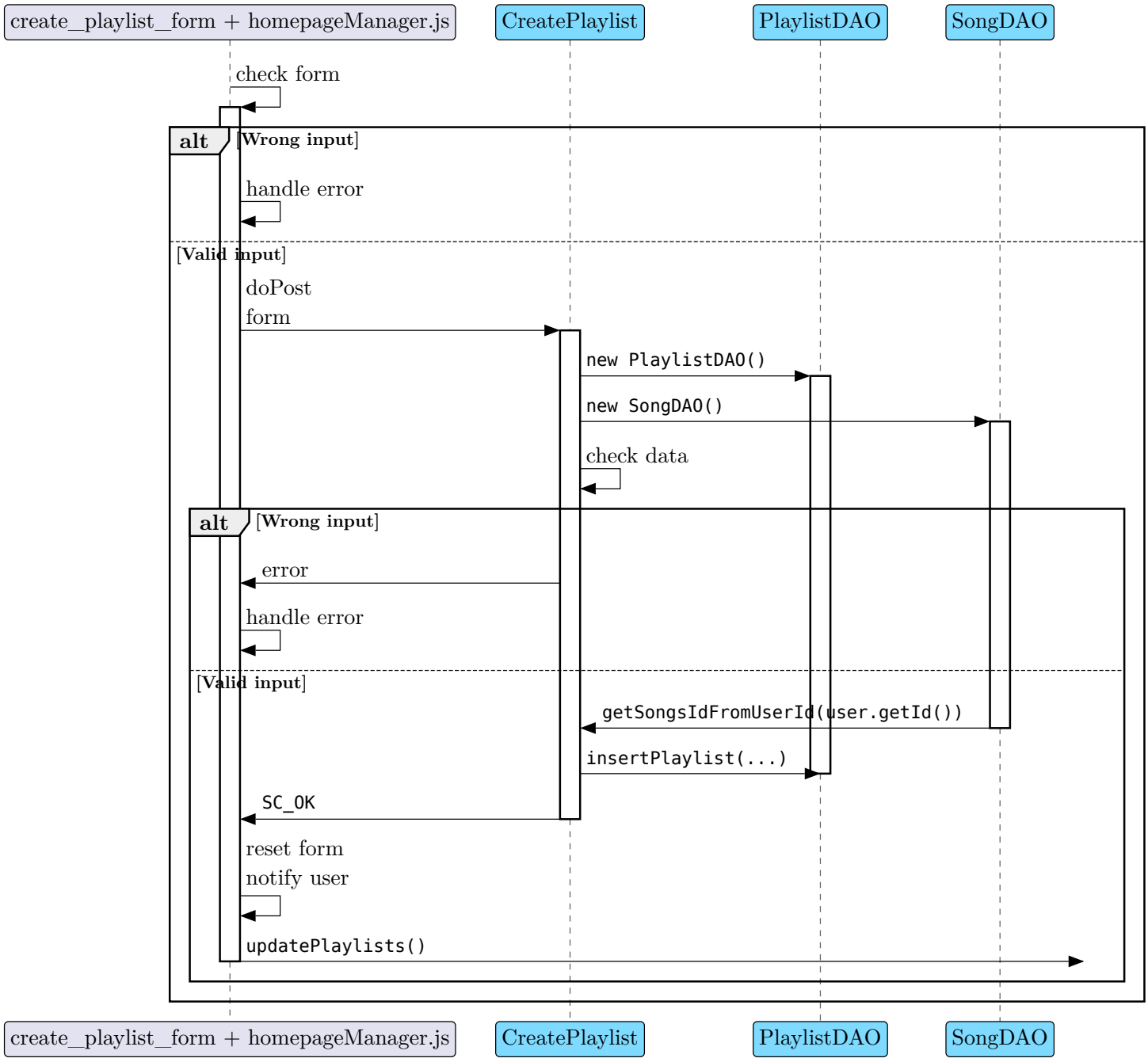
Caricare una canzone



Quando l'utente manda l'`upload_song_form`, controlla tutti i campi e, in caso d'errore, mostra un messaggio di errore e annulla l'operazione. Se tutti i campi sono stati compilati correttamente, fa una chiamata alla servlet di `UploadSong`: se la canzone viene caricata correttamente, resetta il form

e ricarica il form per creare una playlist `updateCreatePlaylistForm()`, altrimenti mostra un messaggio d'errore.

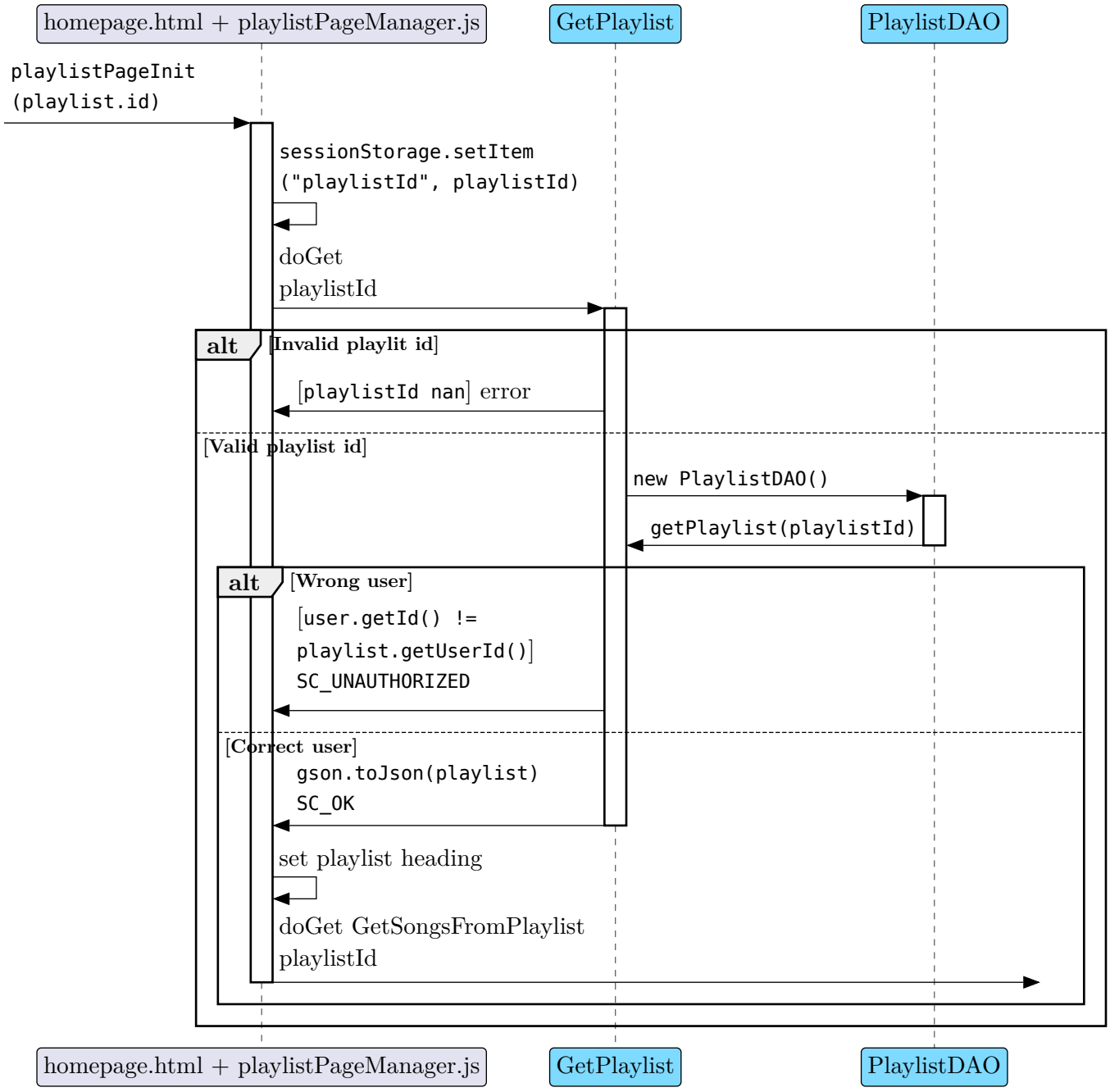
Creare una playlist



Quando l'utente manda il `create_playlist_form`, controlla il campo del titolo e, in caso d'errore, mostra un messaggio di errore e annulla l'operazione. Altrimenti, fa una chiamata alla servlet `CreatePlaylist`: se la playlist viene creata correttamente, resetta il form e ricarica la lista di playlist `updatePlaylists()`, altrimenti mostra un messaggio di errore.

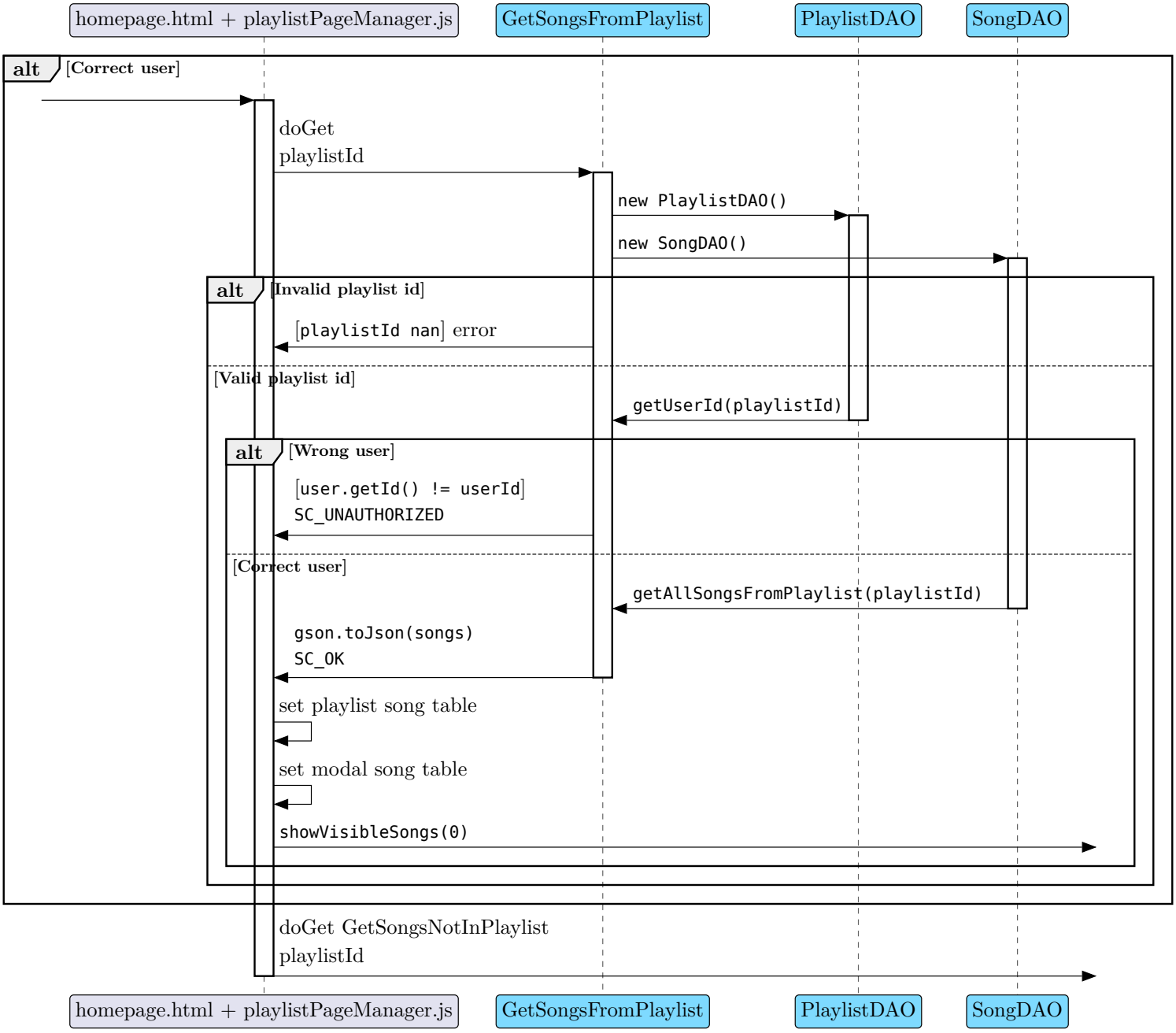
Inizializzare la pagina della playlist

- Parte 1



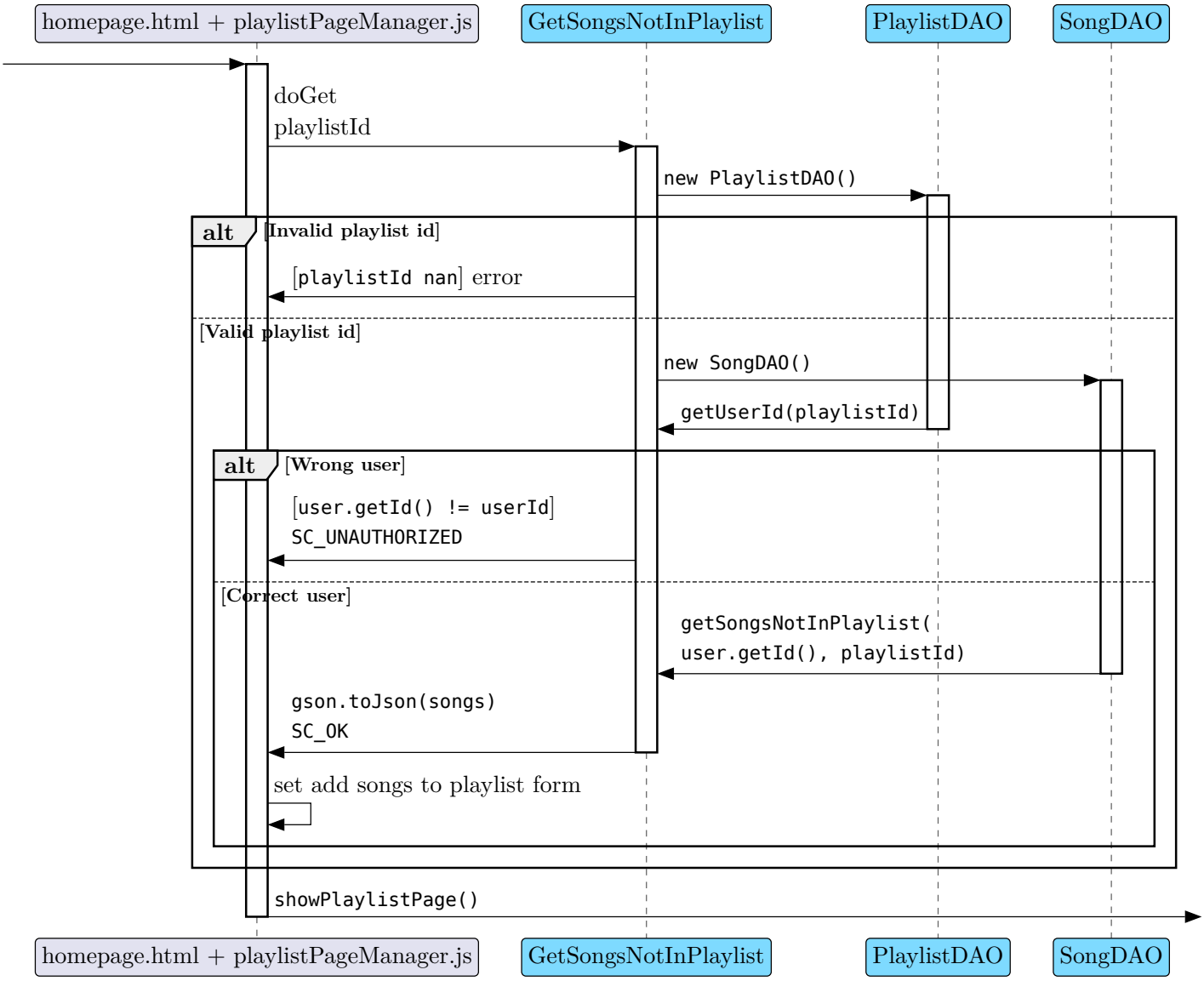
Quando è necessario ricaricare la pagina della playlist, viene chiamata la servlet `GetPlaylist` con il playlistId preso dalla session storage. Se la chiamata ha esito positivo, viene aggiornata la «heading» della pagina con le informazioni della playlist corrente.

- Parte 2



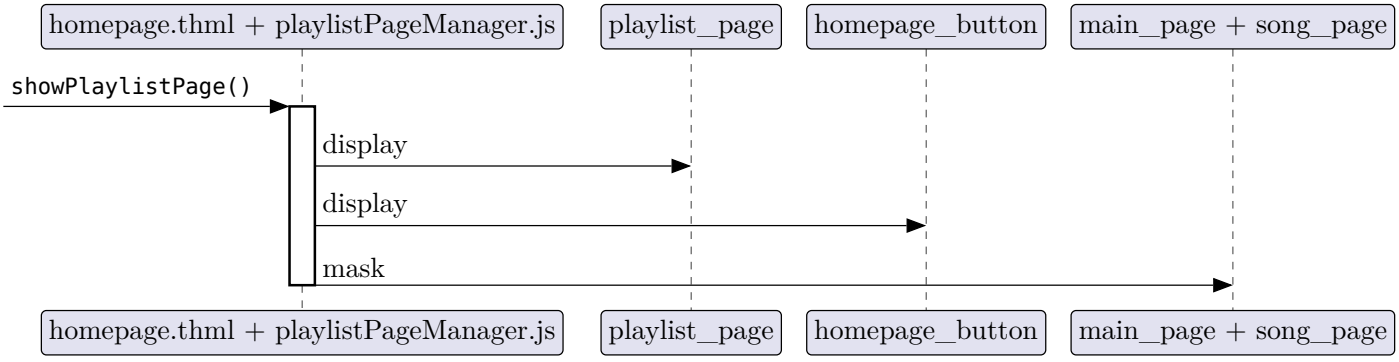
Se la chiamata precedente ha avuto esito positivo, viene chiamata la servlet `GetSongsFromPlaylist`. Se anche questa chiamata ha esito positivo, le canzoni vengono inserite in una tabella in gruppi di 5 e viene chiamata `showVisibleSongs()` per renderizzare correttamente solo il primo gruppo con i corrispettivi pulsanti. Lo stesso JSON viene usato anche per aggiornare il modal.

- Parte 3

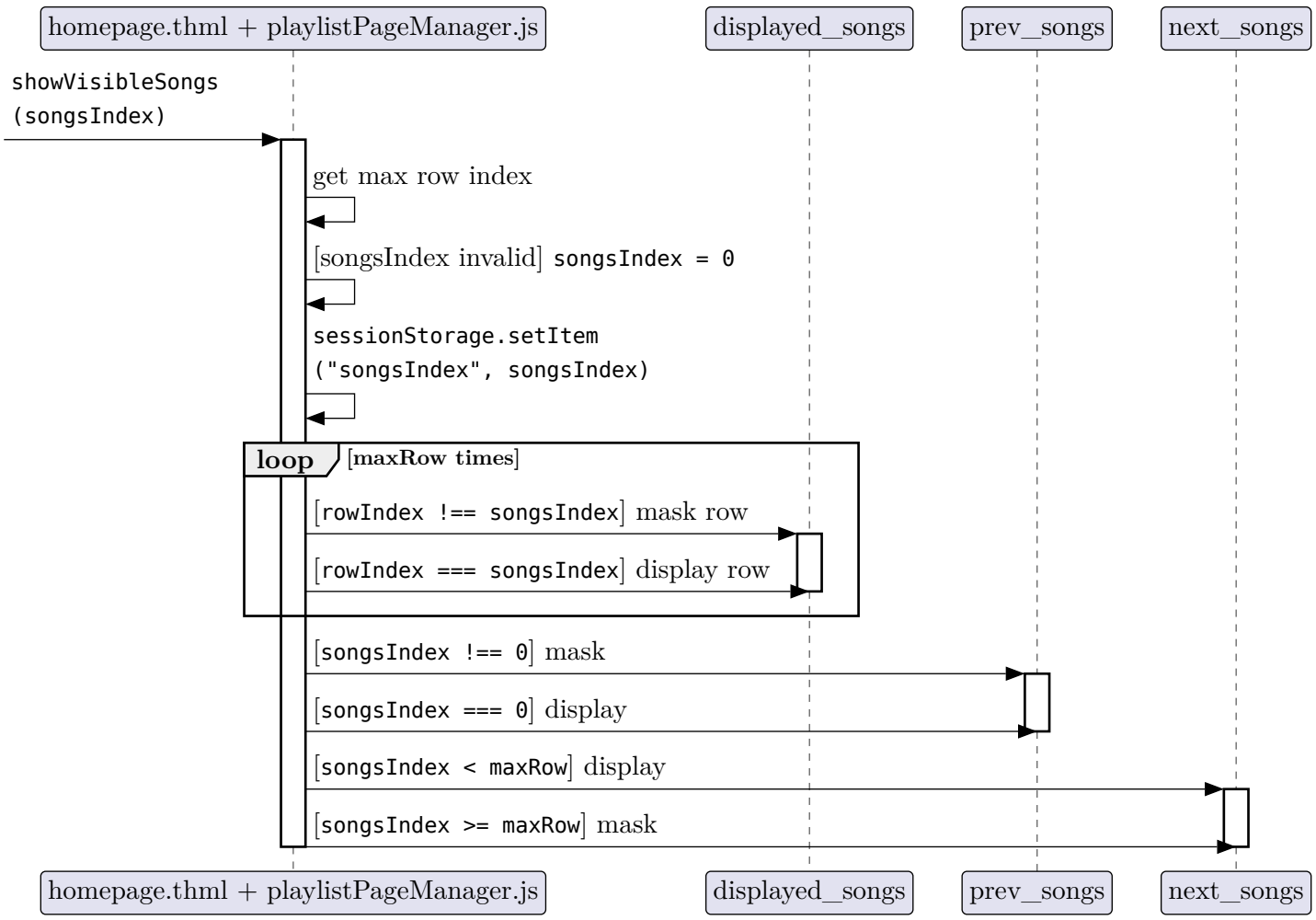


Dopo i passi precedenti, viene chiamata la servlet `GetSongsNotInPlaylist`: se questa chiamata ha esito positivo, viene inizializzato il form per aggiungere canzoni alla playlist corrente con i valori del json. Infine, viene renderizzata la pagina della playlist.

- **Andare alla pagina della playlist**

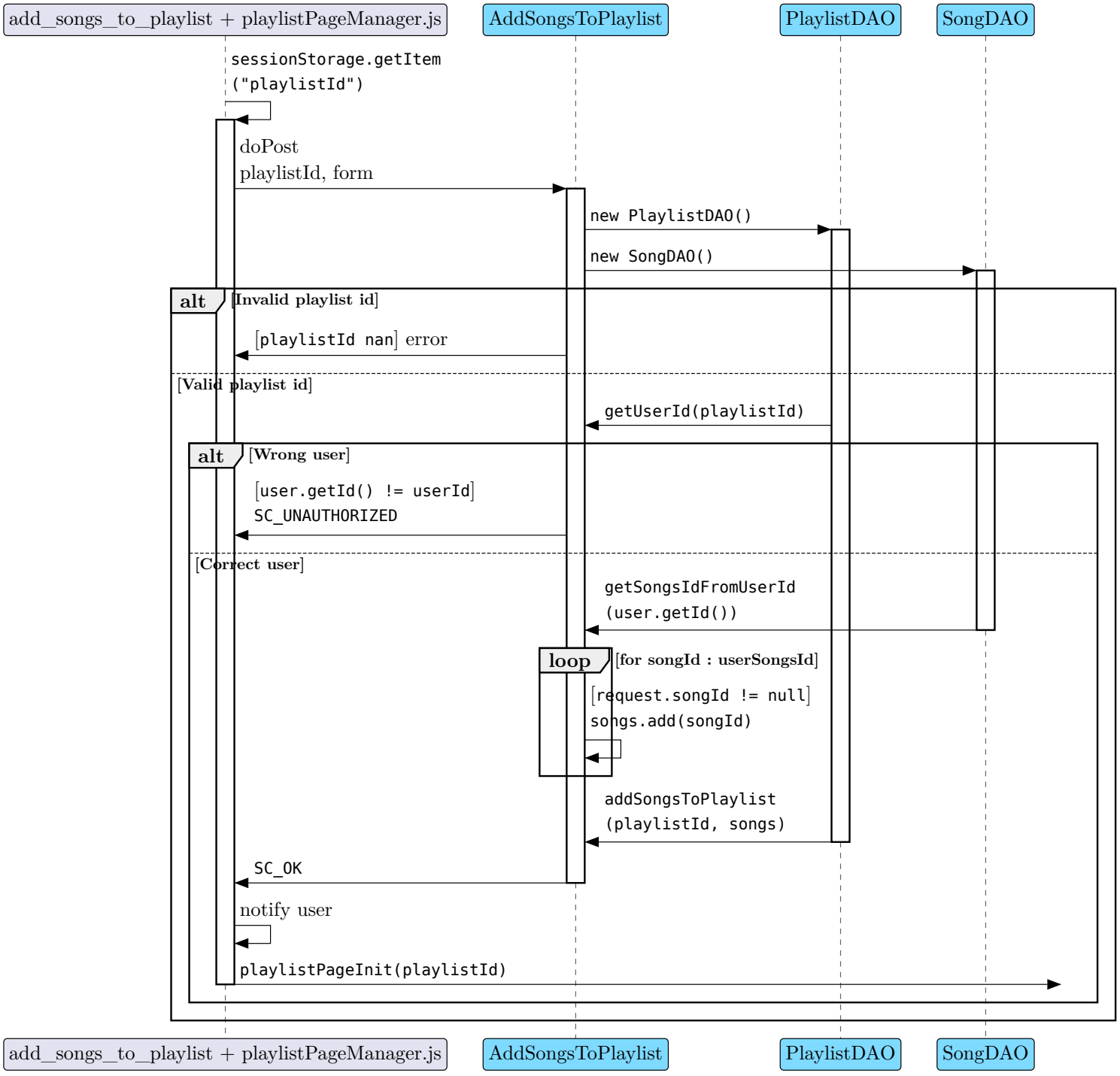


- **Visualizzare brani precedenti/successivi**



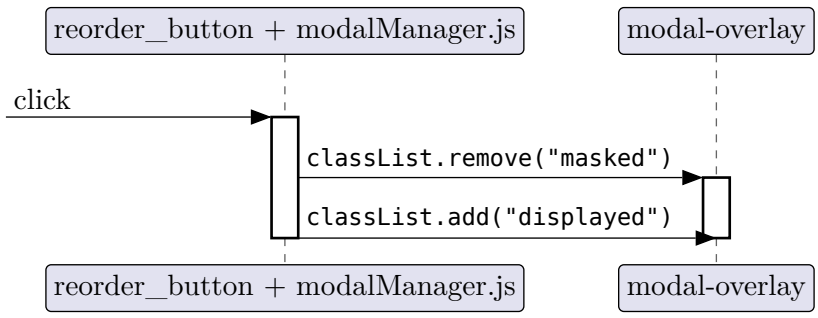
Quando viene inizializzata la pagina della playlist o l'utente clicca i pulsanti «precedenti»/«successivi», la funzione calcola l'indice massimo e controlla che l'indice passato sia valido (numero positivo minore dell'indice massimo): se non lo è, gli viene assegnato il valore di default 0. Il nuovo indice viene poi salvato nella session storage e vengono mascherate tutte le righe il cui indice non coincide con il `songsIndex`, e viene rivelata la riga con lo stesso indice. Infine, in base all'indice, vengono mascherati/rivelati i relativi pulsanti.

- **Aggiungere brani alla playlist**

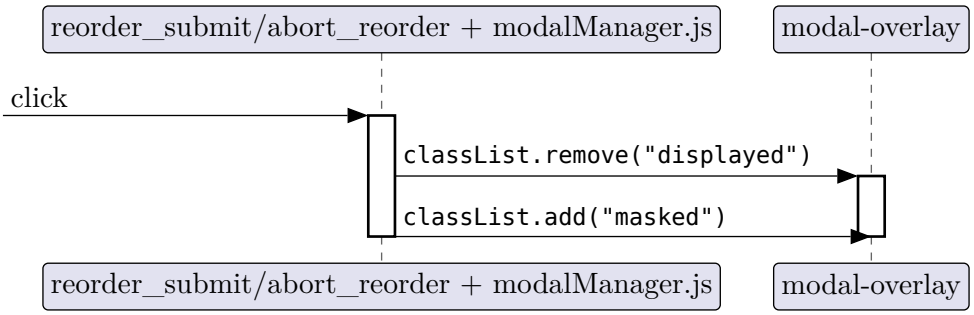


Quando l'utente compila il form, viene controllato che abbia selezionato almeno una canzone: in tal caso, viene chiamata la servlet `AddSongsToPlaylist`. Il resto è simile alla **versione precedente**.

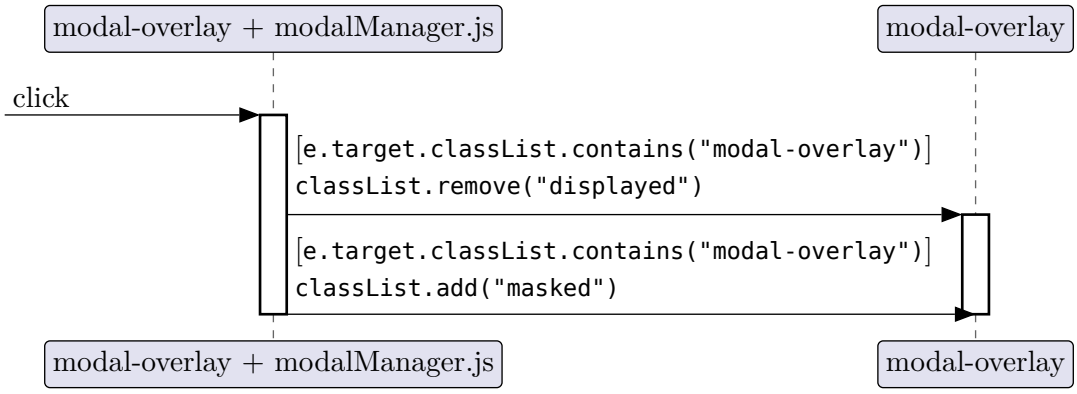
- **Aprire il modal**



- **Chiudere il modal**

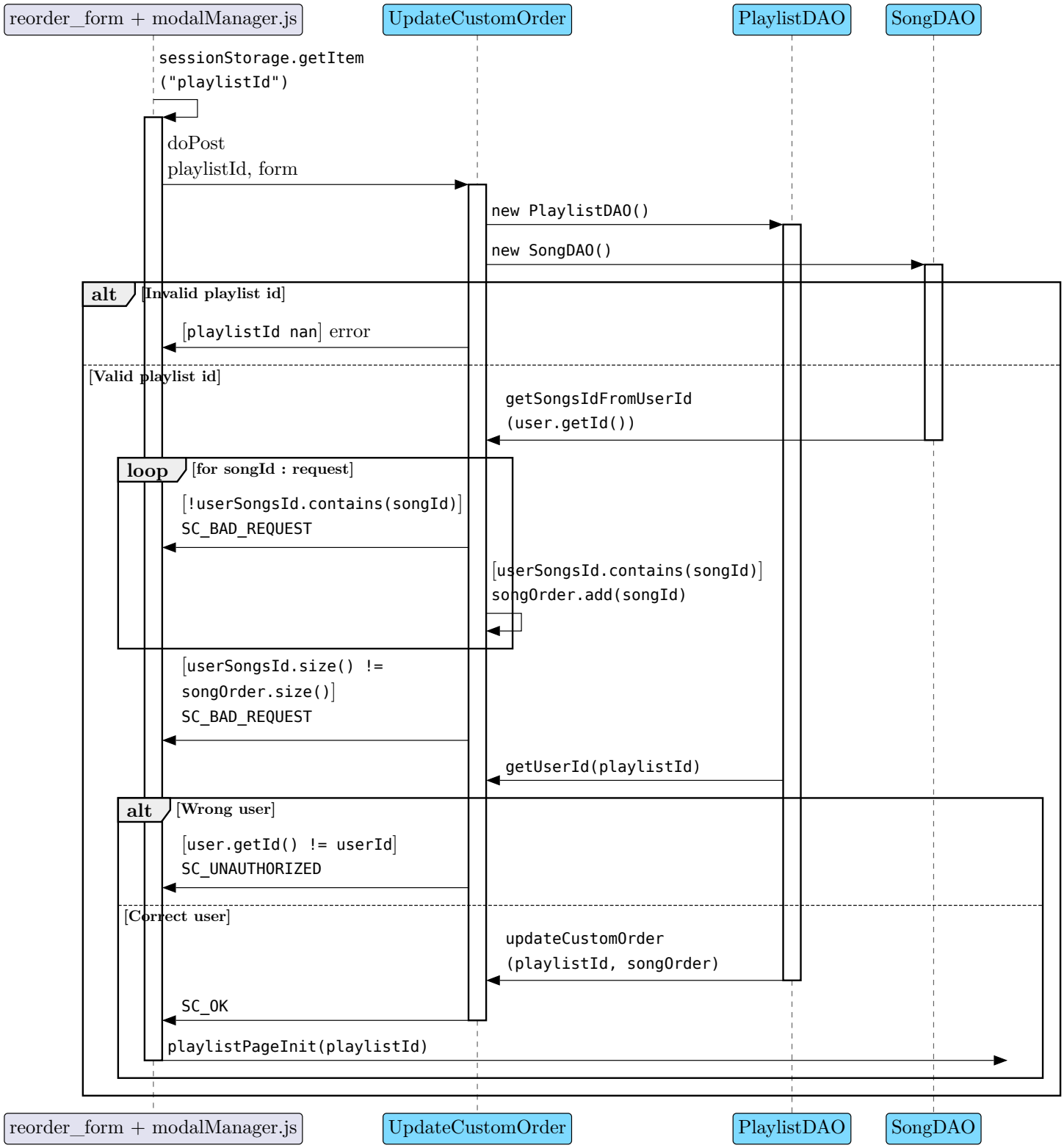


- **Chiudere il modal cliccando fuori dal form**



Simile al diagramma di **Chiudere il modal**, con l'aggiunta di controllare che l'utente abbia effettivamente cliccato il modal (cioè lo sfondo trasparente a lato del modal) e non uno dei contenuti del modal.

Cambiare ordinamento della playlist

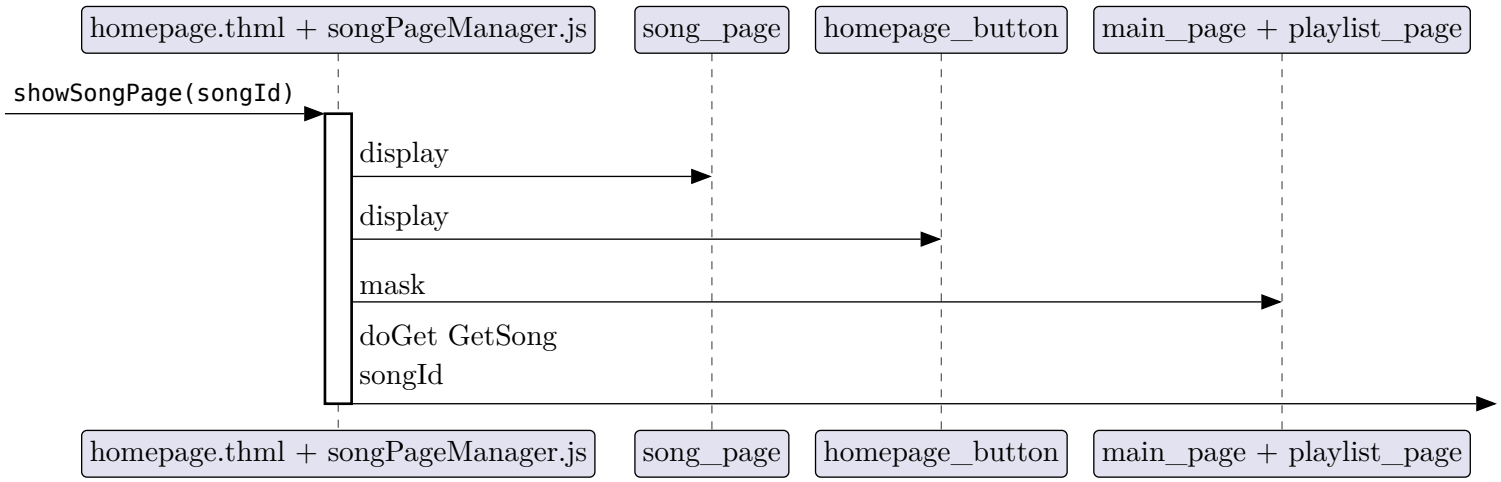


Quando l'utente è soddisfatto con il nuovo ordinamento, viene chiamata la servlet `UploadCustomOrder`, che controlla che il playlist id sia valido e che l'id referenzi una playlist creata dall'utente corrente. Dato che l'ordine delle canzoni in questo caso è importante, viene controllato

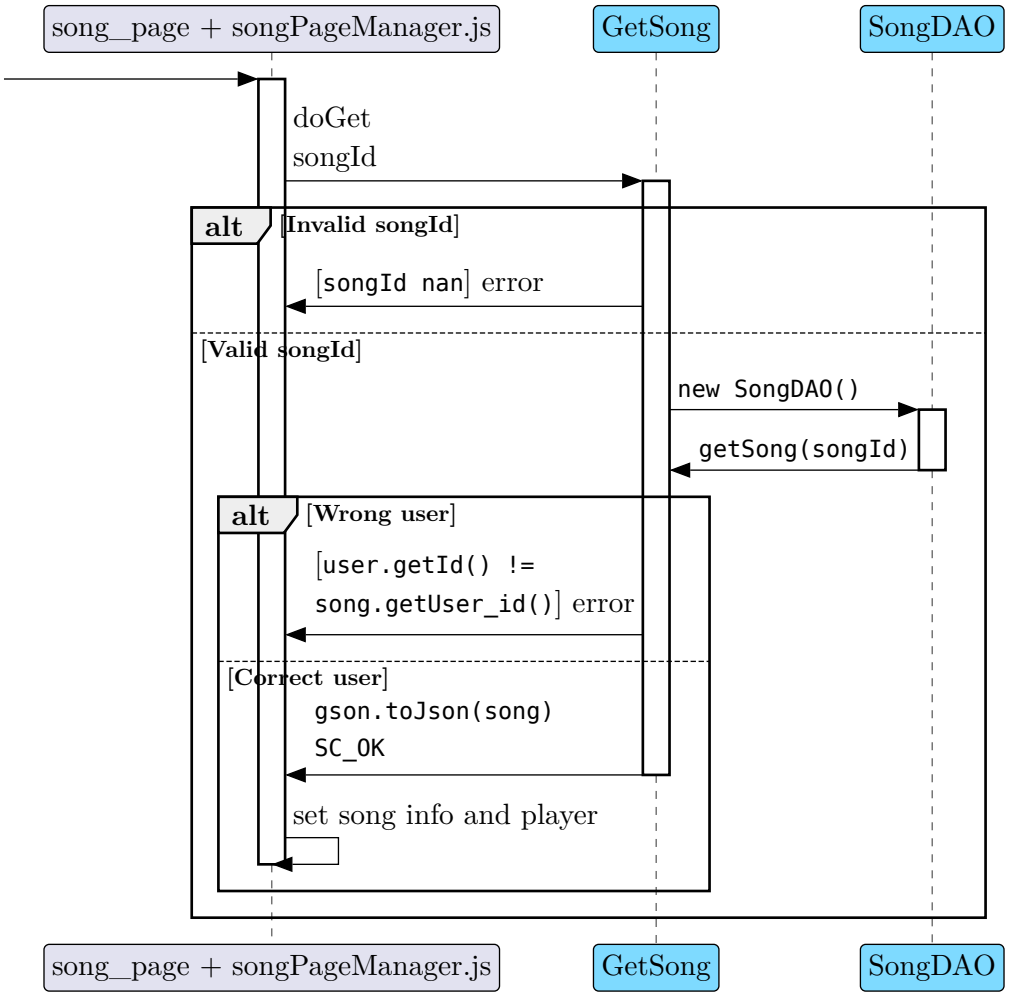
anche che ogni singolo id mandato appartenga alla playlist data, e che la dimensione della lista con il nuovo ordine e la lista dei brani della playlist siano uguali. Se tutti questi controlli hanno successo, viene aggiornato l'ordine personalizzato e viene «ricaricata» la pagina della playlist per renderizzarla col nuovo ordine.

Andare alla pagina della canzone

- Parte 1



- Parte 2



Quando l'utente clicca sul titolo di un brano, viene chiamata la servlet `GetSong`, che recupera la canzone e mostra le sue informazioni.