PLAYLIST MUSICALE

Emir Velicanin

Prof. Fraternali Piero

Anno Accademico 2024-2025

Specifica

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 e 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.

Da specifica a schema concettuale

Ritagliamo dalla specifica le informazioni utili alla progettazione della base di dati per la versione
“pure HTML”

> “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.”
> “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, 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.”

Da specifica a schema concettuale

Suddividiamo le parti utili alla progettazione della specifica in:

- entità e relativi attributi;
- relazioni.

"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."
"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, 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."

Da specifica a schema concettuale

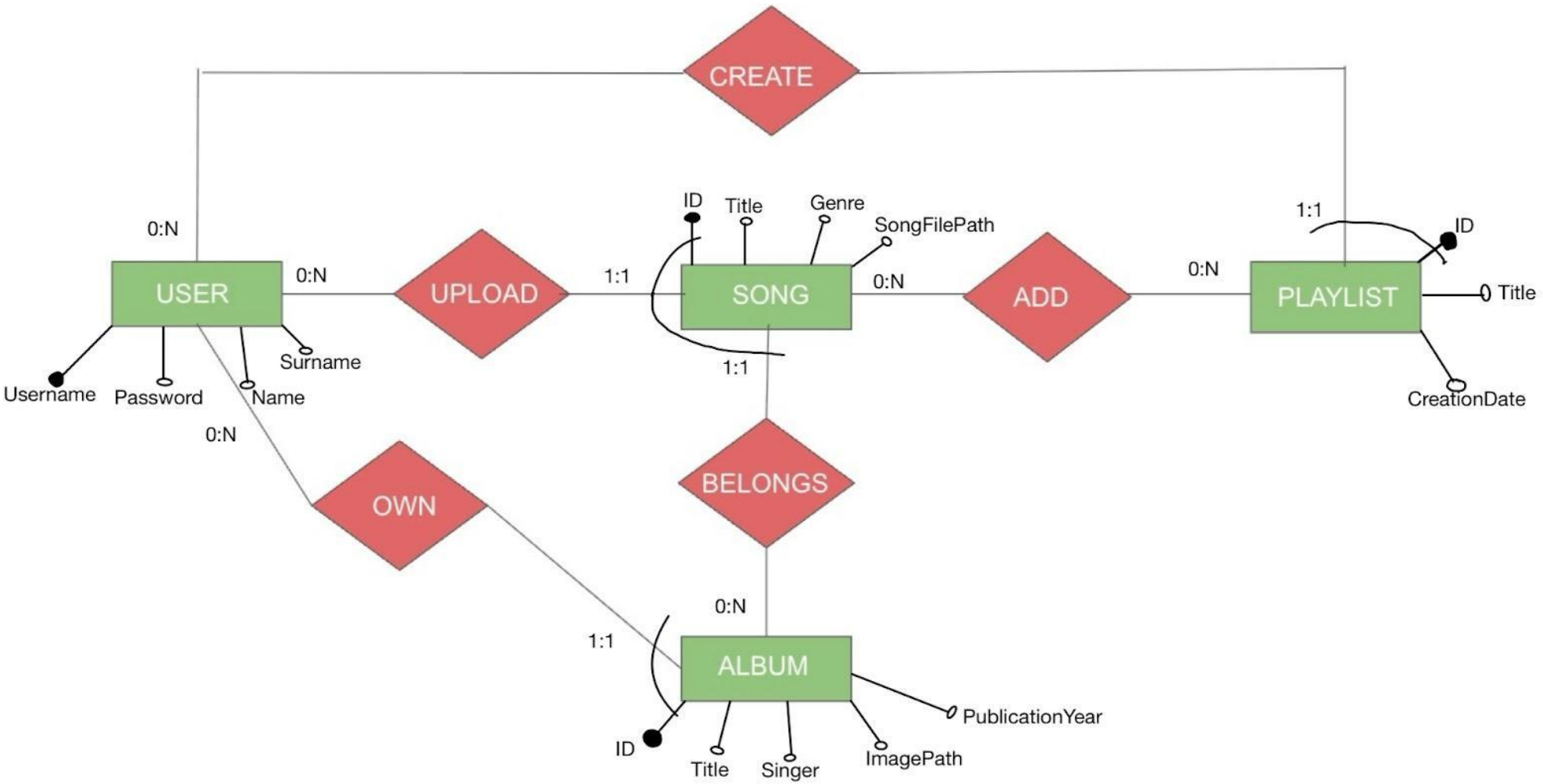
Identifichiamo colorando in **blu** e in **grassetto** gli attributi scelti come chiave primaria):

- entità:
  - "USER" : **Username**, Password, Name, Surname;
  - "SONG" : **ID**, Title, Genre, SongFilePath [**il titolo è univoco tra i brani di un user**]
  - "ALBUM":**ID**, Title, Singer, ImagePath, PublicationYear[**titolo dell'album è univoco tra i brani di un user**]
  - "PLAYLIST" : **ID**, Title, CreationDate [**titolo è univoco tra le playlist di un user**]

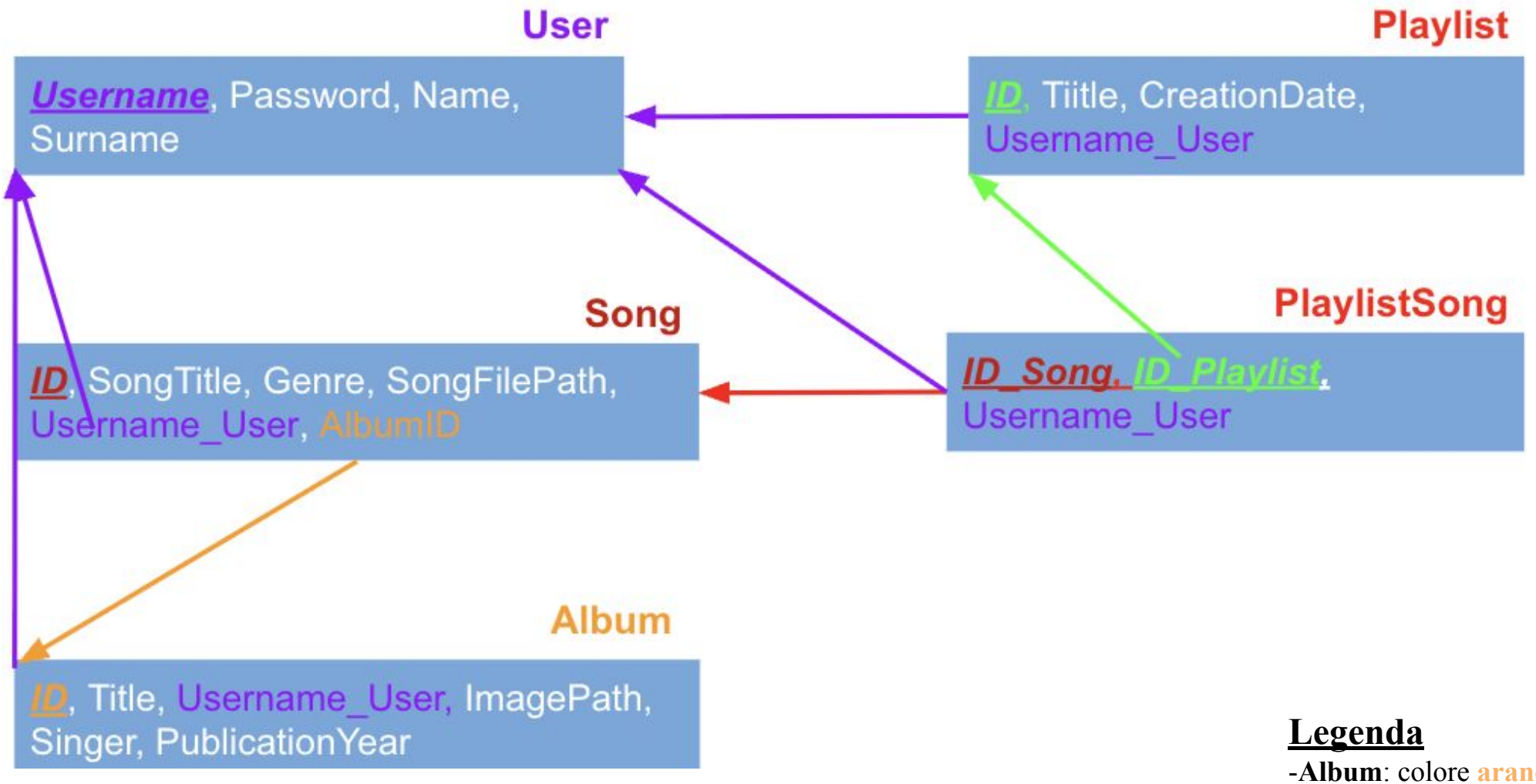
- relazioni:
  - un 'USER' può creare da "0" a "N" 'PLAYLIST' e una 'PLAYLIST' corrisponde uno e un solo 'USER';
  - un 'USER' può caricare da "0" a "N" 'SONG' e una 'song' corrisponde ad uno e un solo 'USER';
  - una 'SONG' appartiene a uno e uno solo 'ALBUM' e un 'ALBUM' contiene da "0" ad "N" 'SONG';
  - una 'SONG' può essere aggiunto a "0" a "N" 'PLAYLIST' e una 'PLAYLIST' può contenere "0" a "N" 'SONG'

Abbiamo ora tutte le informazioni per realizzare lo schema ER della base di dati.

SCHEMA ER


CREATE
0:N
USER
0:N
UPLOAD
1:1
SONG
0:N
ADD
0:N
PLAYLIST
1:1
ID
Title
Genre
SongFilePath
ID
Title
Username
Password
Name
Surname
1:1
CreationDate
0:N
OWN
BELONGS
0:N
1:1
ALBUM
PublicationYear
ID
Title
Singer
ImagePath

Da schema concettuale a schema logico



Legenda

-**Album**: colore arancio
-**Playlist**: colore verde
-**Song: colore rosso**
-**User: colore viola**

Schema Database:USER

```
CREATE TABLE USER(
        Username VARCHAR(50) NOT NULL, AUTO INCREMENT ,
        AUTO_INCREMENT,
        Password VARCHAR(50) NOT NULL,
        Name VARCHAR(50) NOT NULL,
        Surname VARCHAR(50) NOT NULL,
        PRIMARY KEY(Username)
};
```

Schema Database:PLAYLIST

```
CREATE TABLE PLAYLIST(
    ID int NOT NULL AUTO_INCREMENT,
    Title VARCHAR(50) NOT NULL,
    creationDate DATE NOT NULL,
    Username_User VARCHAR(50) NOT NULL,
    PRIMARY KEY (ID),
    FOREIGN KEY(Username_User) REFERENCES USER(username)
    ON DELETE CASCADE ON UPDATE CASCADE)
    };
```

Schema Database:SONG

```
CREATE TABLE SONG(
    ID int NOT NULL AUTO_INCREMENT,
    SongTitle VARCHAR(50) NOT NULL,
    AlbumID INT NOT NULL,
    SongFilePath LONGLOB NOT NULL,
    Genre VARCHAR(50) NOT NULL,
    UsernameUser VARCHAR(50) NOT NULL,
    PRIMARY KEY (ID),
    FOREIGN KEY(Username_User) REFERENCES USER(username)
    ON DELETE CASCADE ON UPDATE CASCADE
    FOREIGN KEY(AlbumID) REFERENCES ALBUM(ID)
    ON DELETE CASCADE ON UPDATE CASCADE
    };
```

Schema Database:ALBUM

```
CREATE TABLE ALBUM(
    ID int NOT NULL AUTO_INCREMENT,
    Title VARCHAR(50) NOT NULL,
    Singer VARCHAR(50) NOT NULL,
    ImagePath LONGLOB NOT NULL,
    PublicationYear INT NOT NULL,
    Username_User VARCHAR(50) NOT NULL,
    PRIMARY KEY (ID),
    FOREIGN KEY(Username_User) REFERENCES USER(username)
    ON DELETE CASCADE ON UPDATE CASCADE)
    };
```

Schema Database:PLAYLISTSONG

```
CREATE TABLE PLAYLISTSONG(
      ID_Playlist INT NOT NULL,
      ID_Song INT NOT NULL,
      Username_User VARCHAR(50) NOT NULL,
      PRIMARY KEY (ID_Playlist,ID_Song),
      FOREIGN KEY (ID_Song) REFERENCES
      SONG(ID)  ON DELETE CASCADE ON UPDATE
      CASCADE,
      FOREIGN KEY(ID_Playlist) REFERENCES
      PLAYLIST(ID)  ON DELETE CASCADE ON
      UPDATE CASCADE,
      FOREIGN KEY(Username_User) REFERENCES
      USER(username) ON DELETE CASCADE ON
      UPDATE CASCADE)
};
```

Legenda:
- **AGES (VIEWS)**
- **VIEW COMPONENTS**

Analisi dei requisiti -1

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 e 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:
- **EVENTS**
- **ACTIONS**

Analisi dei requisiti - 2

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 e 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 “HTML pure”

Componenti

Model objects (JavaBeans)

- User
- Playlist
- Song
- Album

Data Access Objects

- **UserDAO**
  - createUser(username, password,name,surname)
  - findUser(username)
  - checkAuthentication(username,password)
  - getAllUserSongs(username, getAlbumCovers)
  - findPlaylists(username)
  - getAllAlbums(username)
  - deleteAccount(username)
- **PlaylistDAO**
  - createPlaylist(title, songList, username)
  - findPlaylist(username)
  - checkPlaylistOwner(playlistID, username)
  - getPlaylistTitle(playlistID, username)
  - countSongs(playlistID, username)
  - getSongsNotInPlaylist(username, playlistID)
  - getSongsFiltered(playlistID, username, offset)
  - getSongs(playlistID, username)
  - removeSongFromPlaylist(songID, playlistID, username)
  - deletePlaylist(playlistID, username)
  - uploadSongNotExistingAlbum(songTitle, albumTitle,singer, publicationYear, genre, imagePath,songFilePath, usernameUser)
- **SongDAO**
  - uploadSongExistingAlbum(songTitle, albumID, genre, ,songFilePath, usernameUser)
  - existAlbum(albumTitle, username)
  - addSongToPlaylist(songID, playlistID, username)
  - addSongsToPlaylist(songList, playlistID,username)
  - getSong(songID, username)
  - checkSongOwner(songID, username)
  - deleteSong(songID, username)

Componenti

Controllers(servlets)

- **Authentication**
  - CheckLogin
  - Logout
  - CheckRegistration
  - DeleteAccount
- **Get content**
  - GetAllPlaylist
  - GetAllSong
  - GoToHomePage
- **Create content**
  - AddSongsToPlalylist
  - AddSong
- **Delete content**
  - DeletePlaylist
  - DeleteSong
  - RemoveSongFromPlaylist

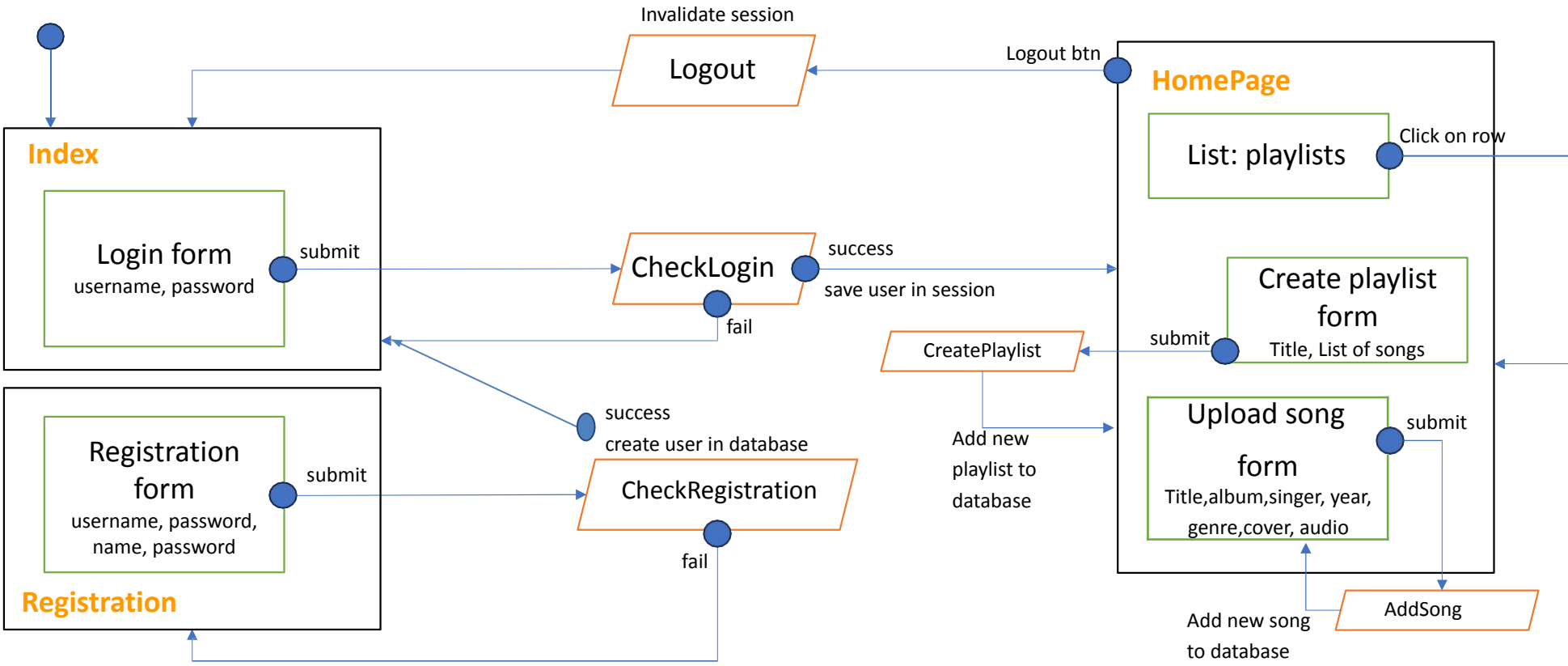
Views(Templates)

- Index(Login)
- Registration
- HomePage
- PlaylistPage
- PlayerPage

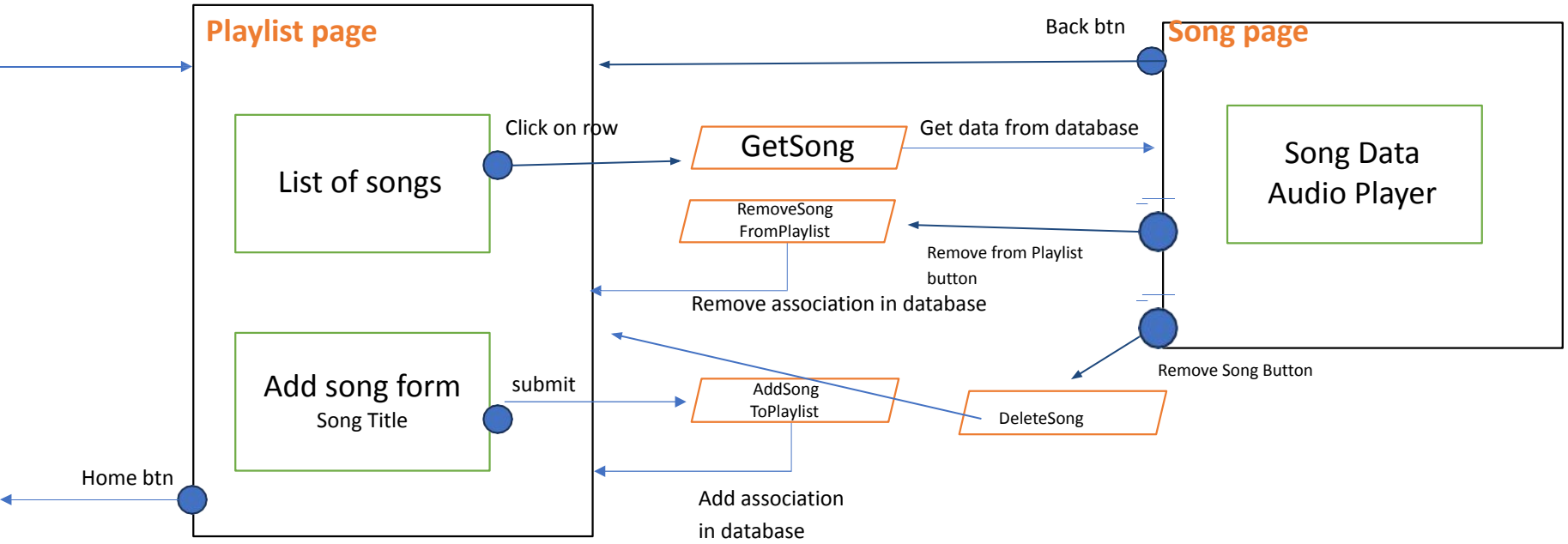
Utilities

- ConnectionHandler

Application design (login/create account)

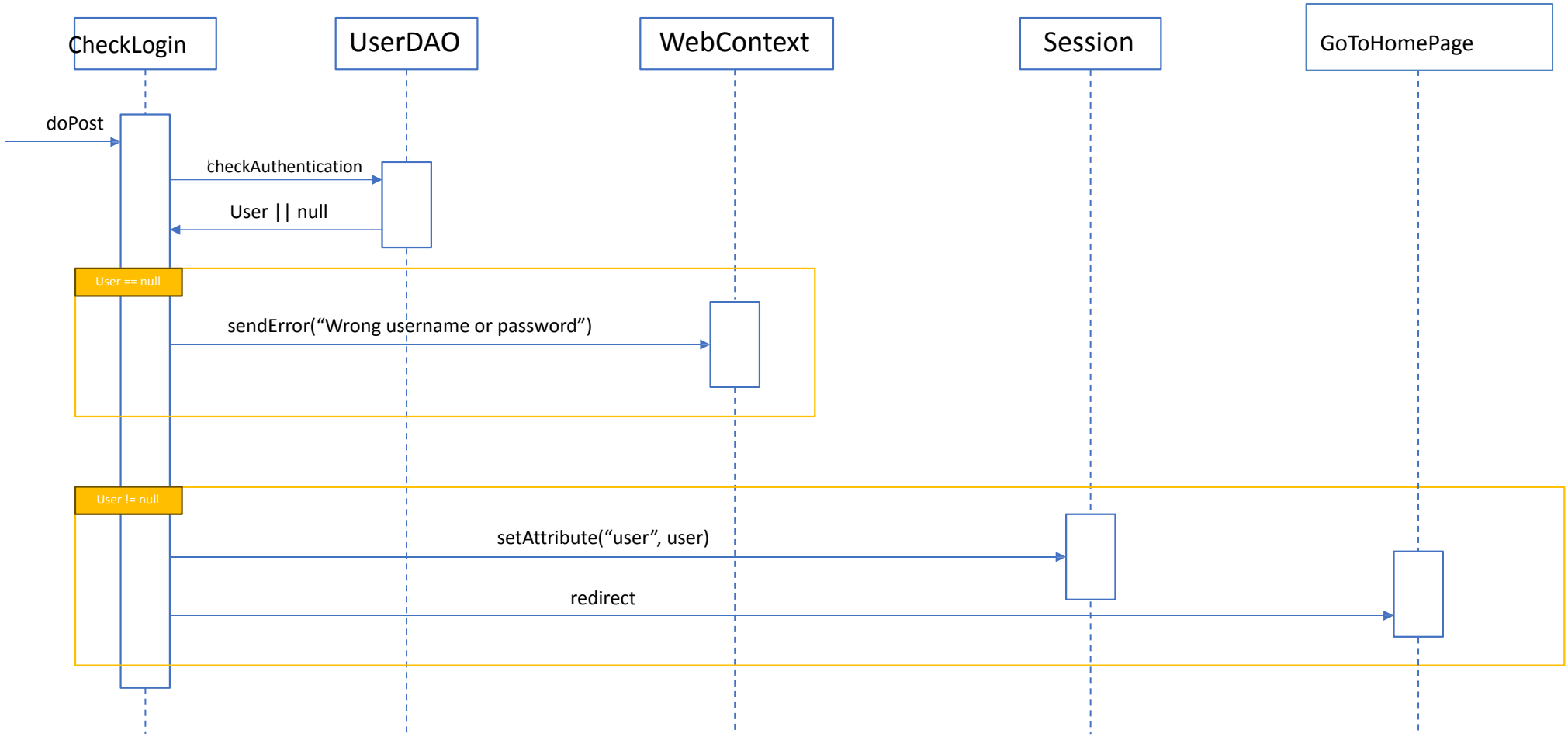


Application design: playlist/song



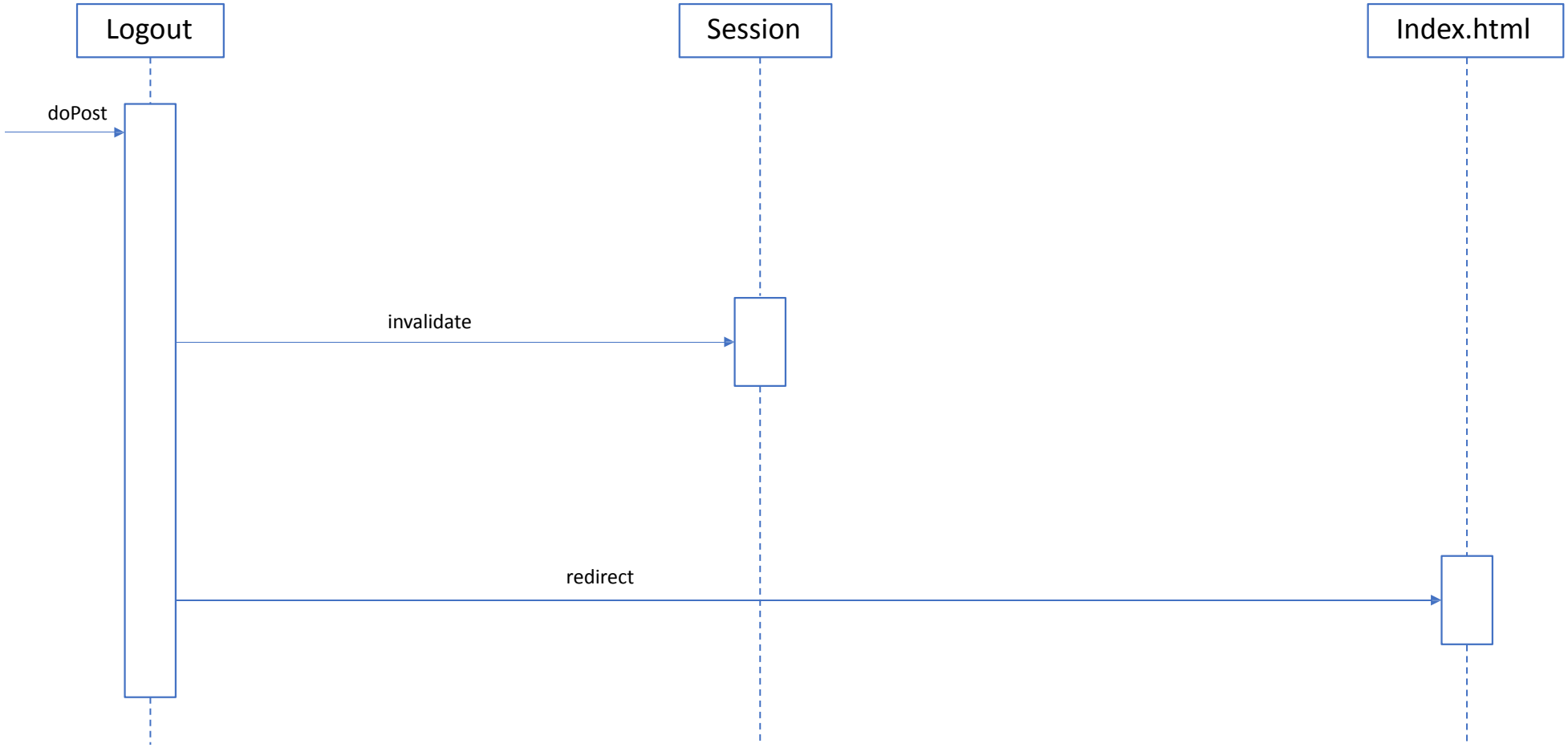
Event: login

POST request from **index.html**
Parameters: username, password



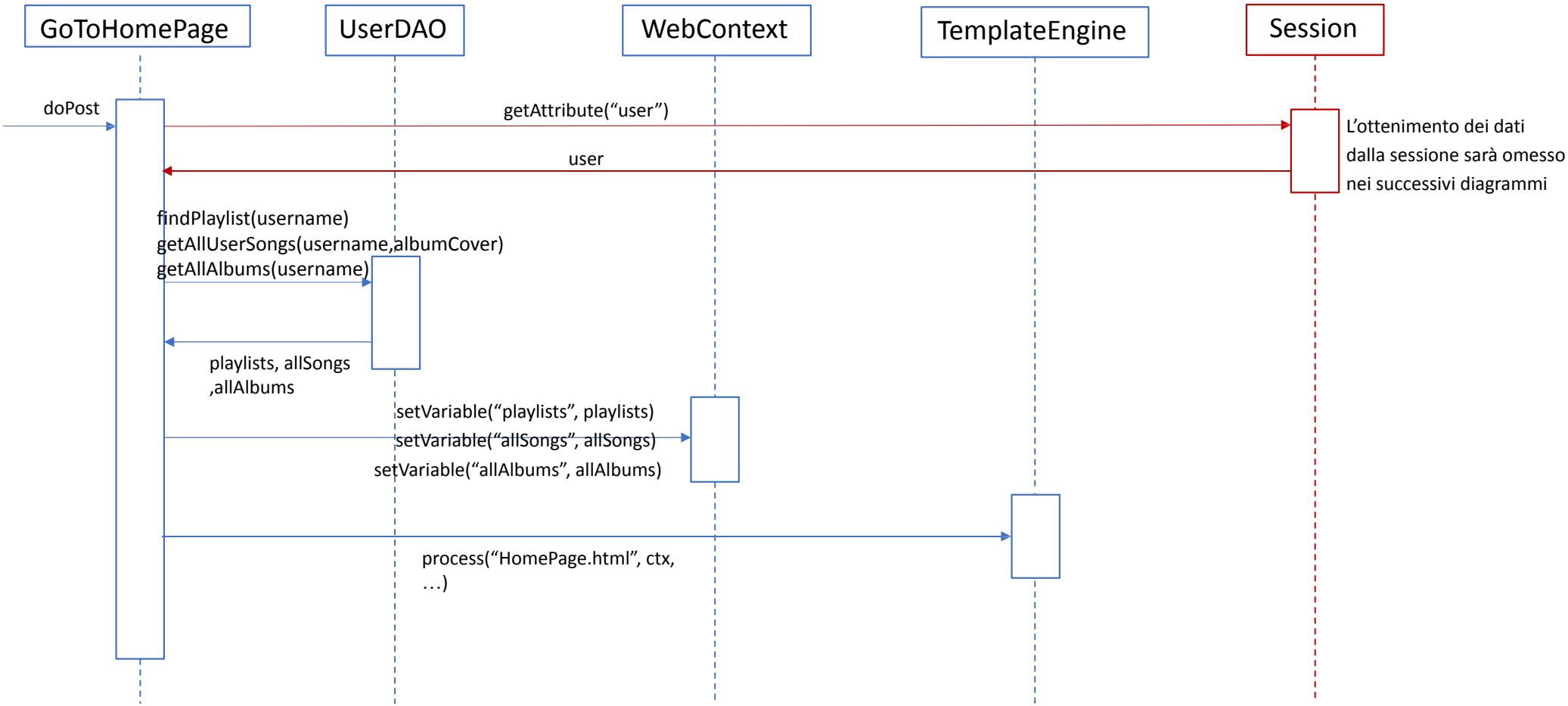
Event: logout

POST request from **HomePage.html**



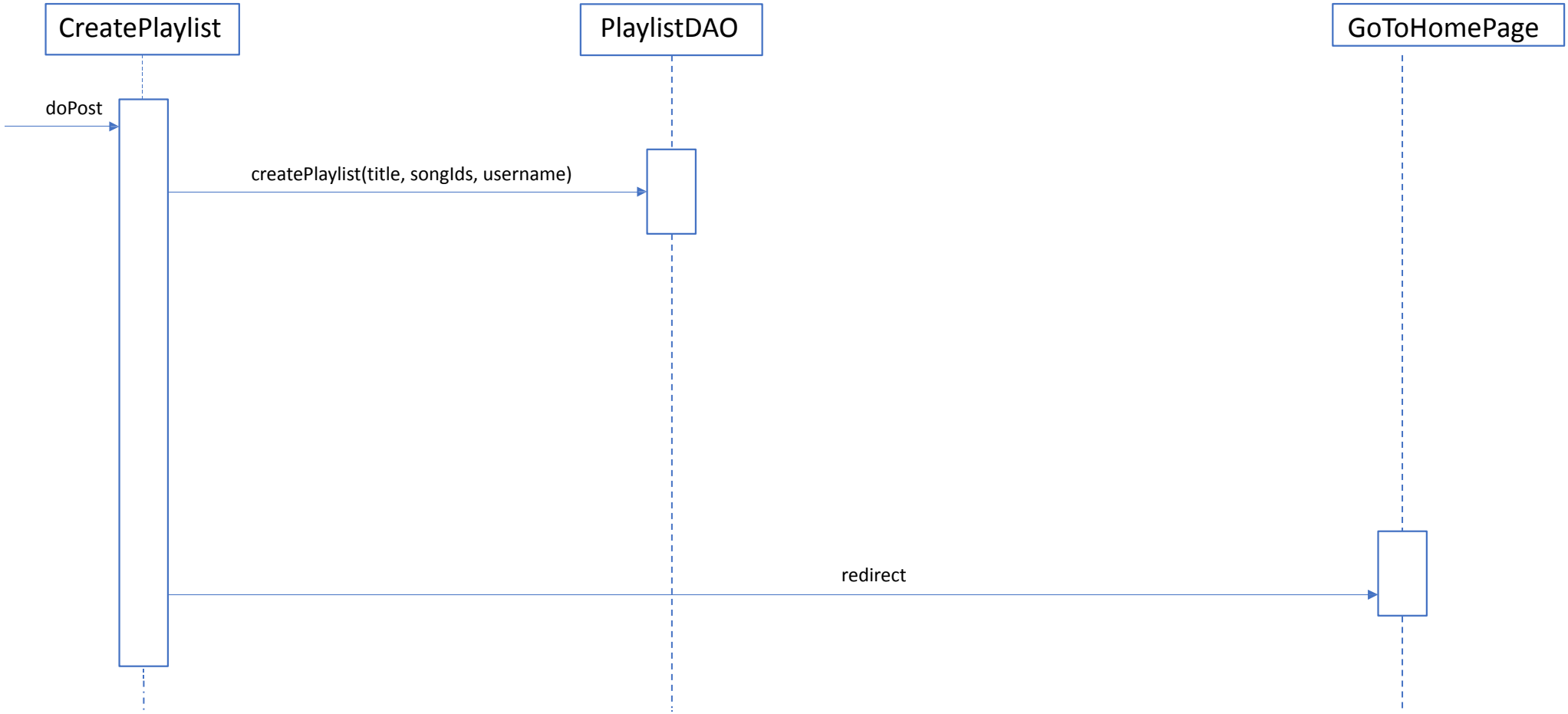
View: GoToHomePage

GET request



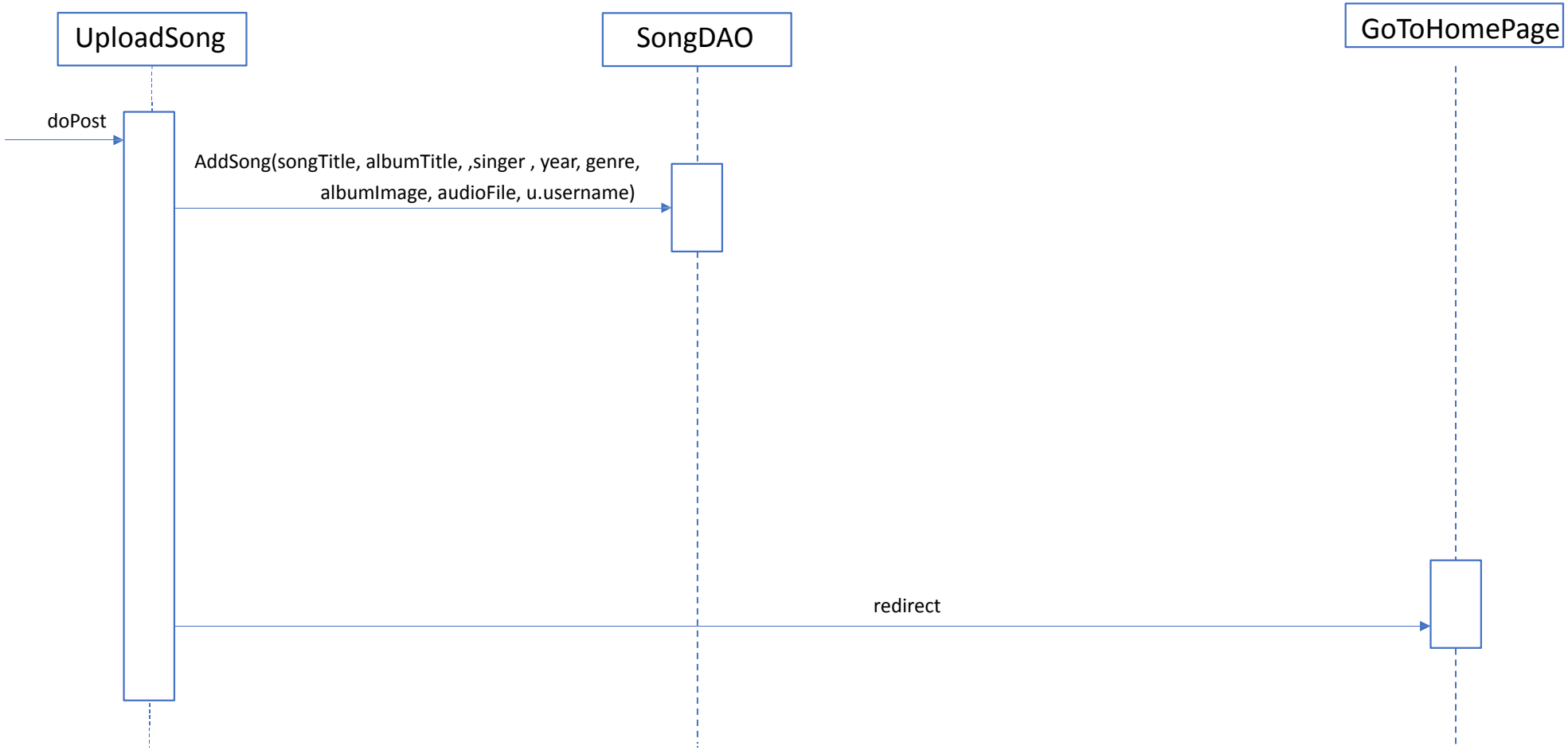
Event: CreatePlaylist

POST request from **Home.html**
Parameters: name, songIds



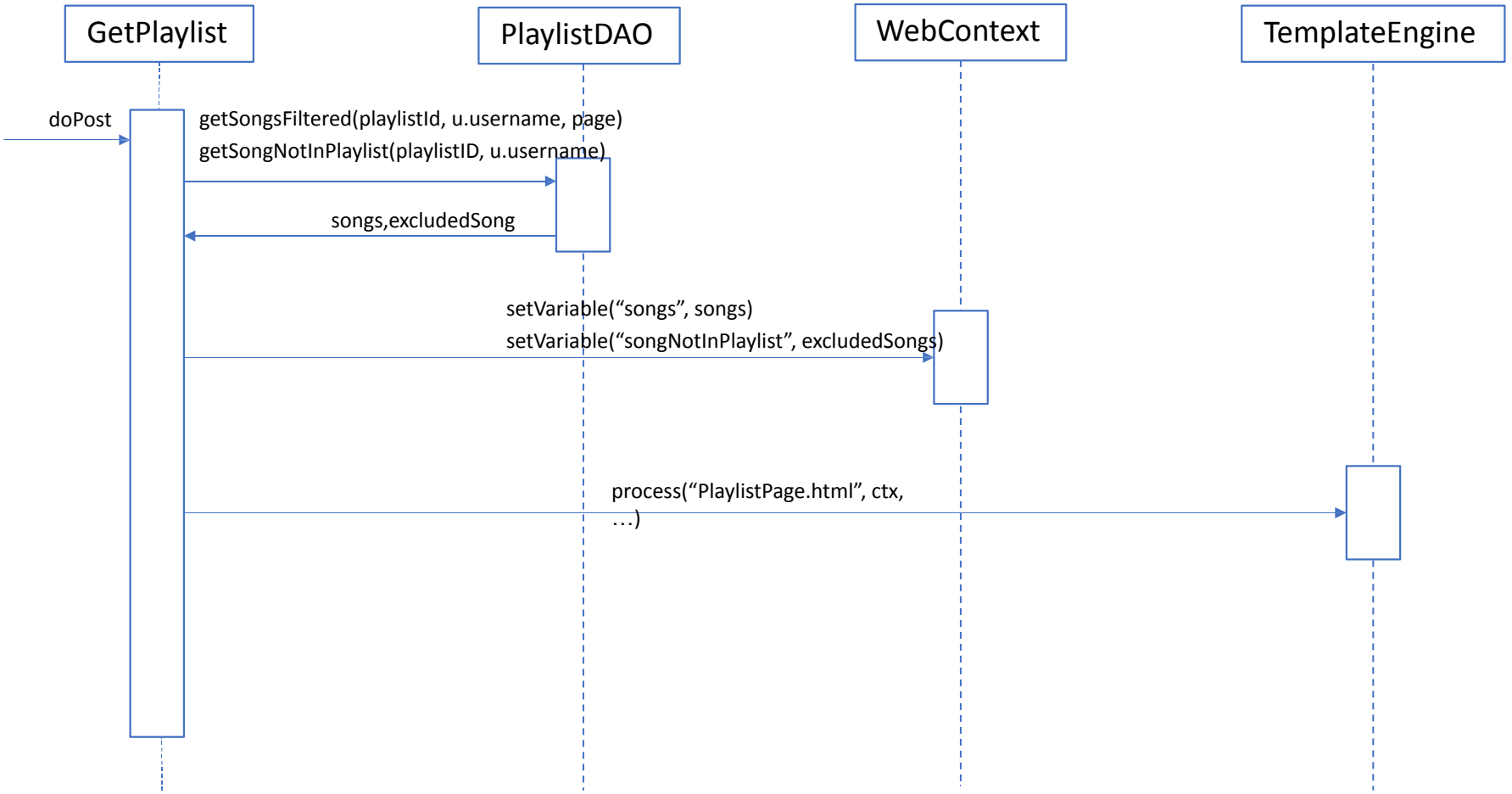
Event: AddSong

POST request from **Home.html**
Parameters: songTitle, albumTitle, singer, year, genre, albumImage, audioFile



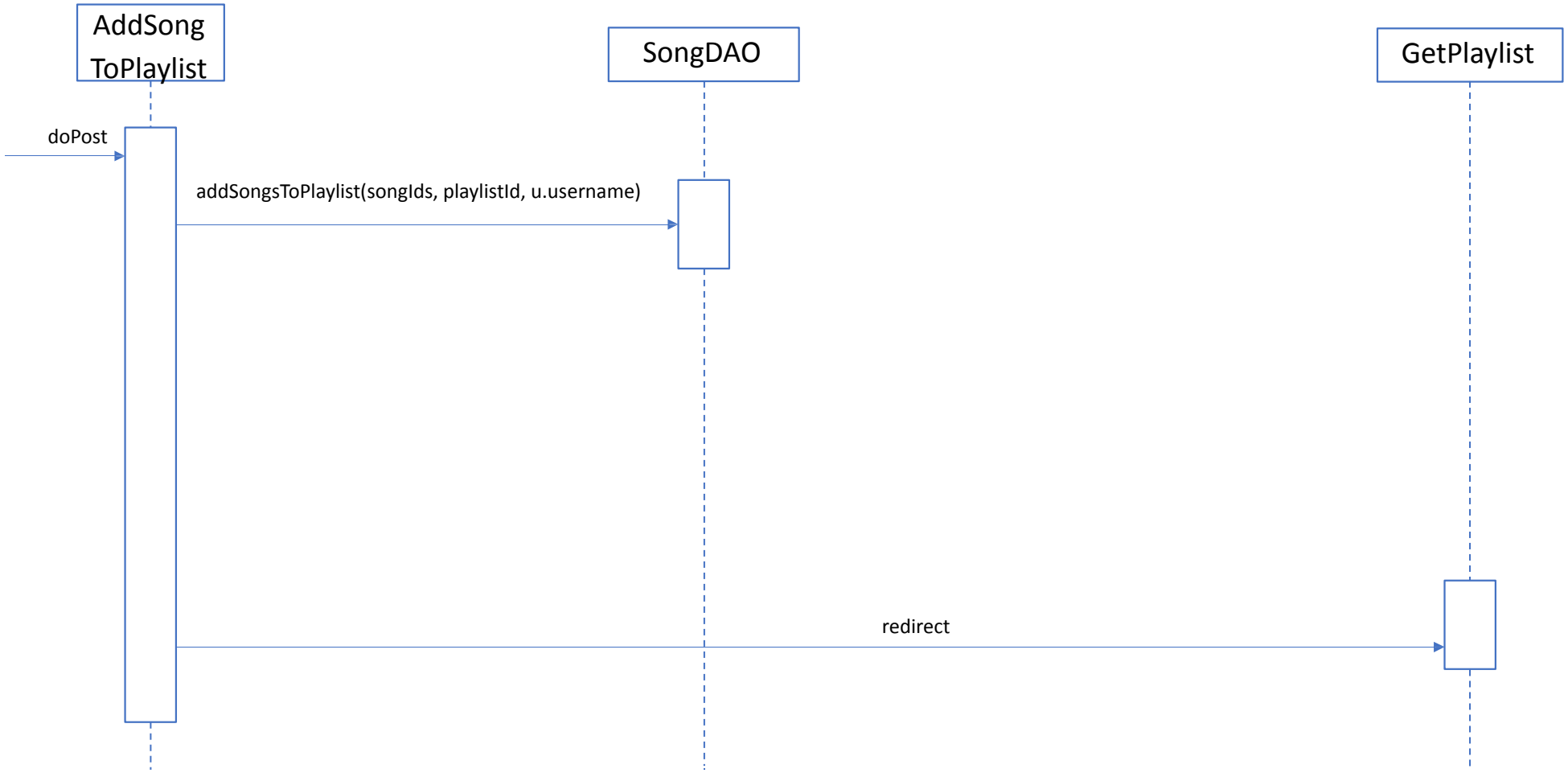
View: getPlaylist

GET request from **HomePage.html**
Parameters: playlistID



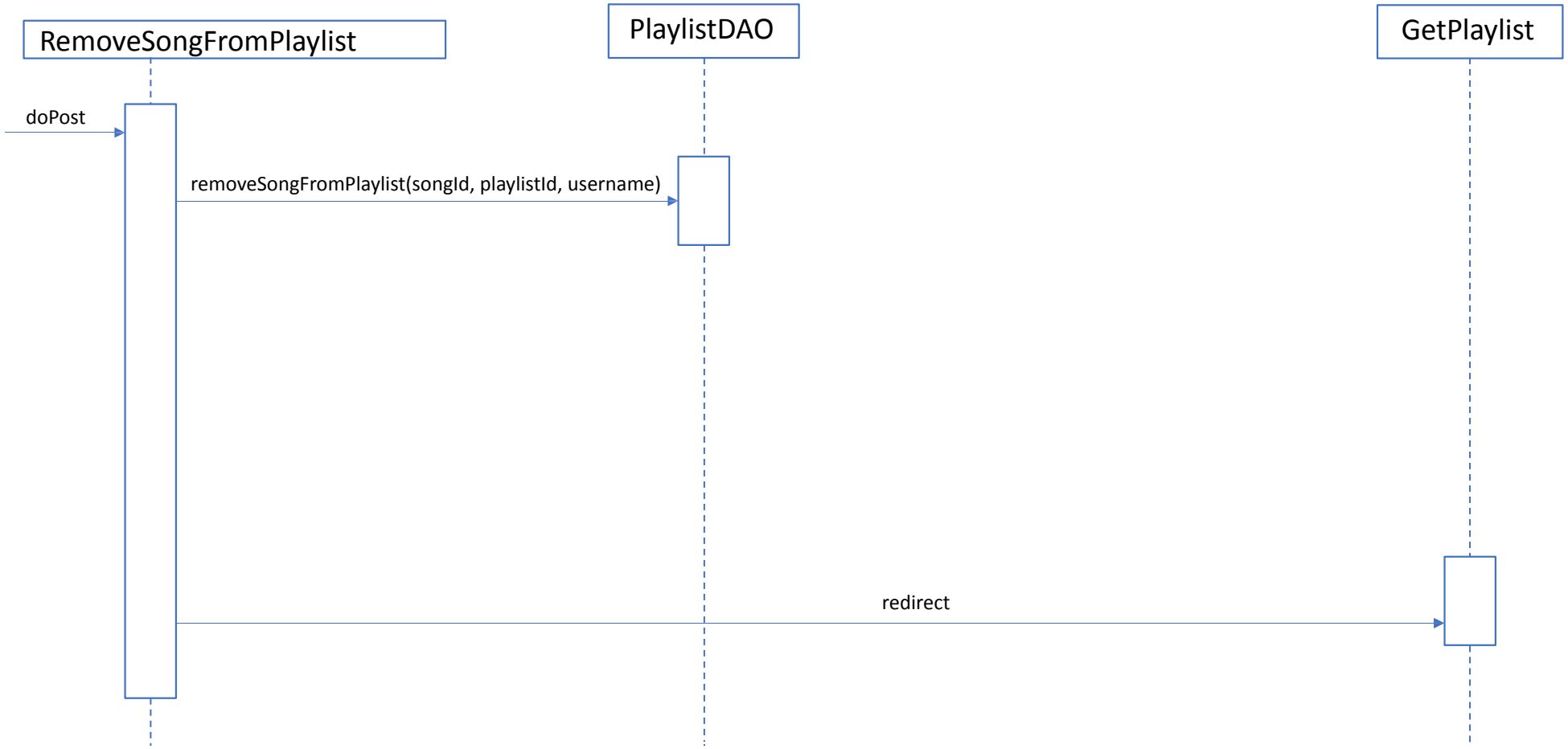
Event: AddSongsToPlaylist

POST request from **PlaylistPage.html**
Parameters: songIds, playlistID



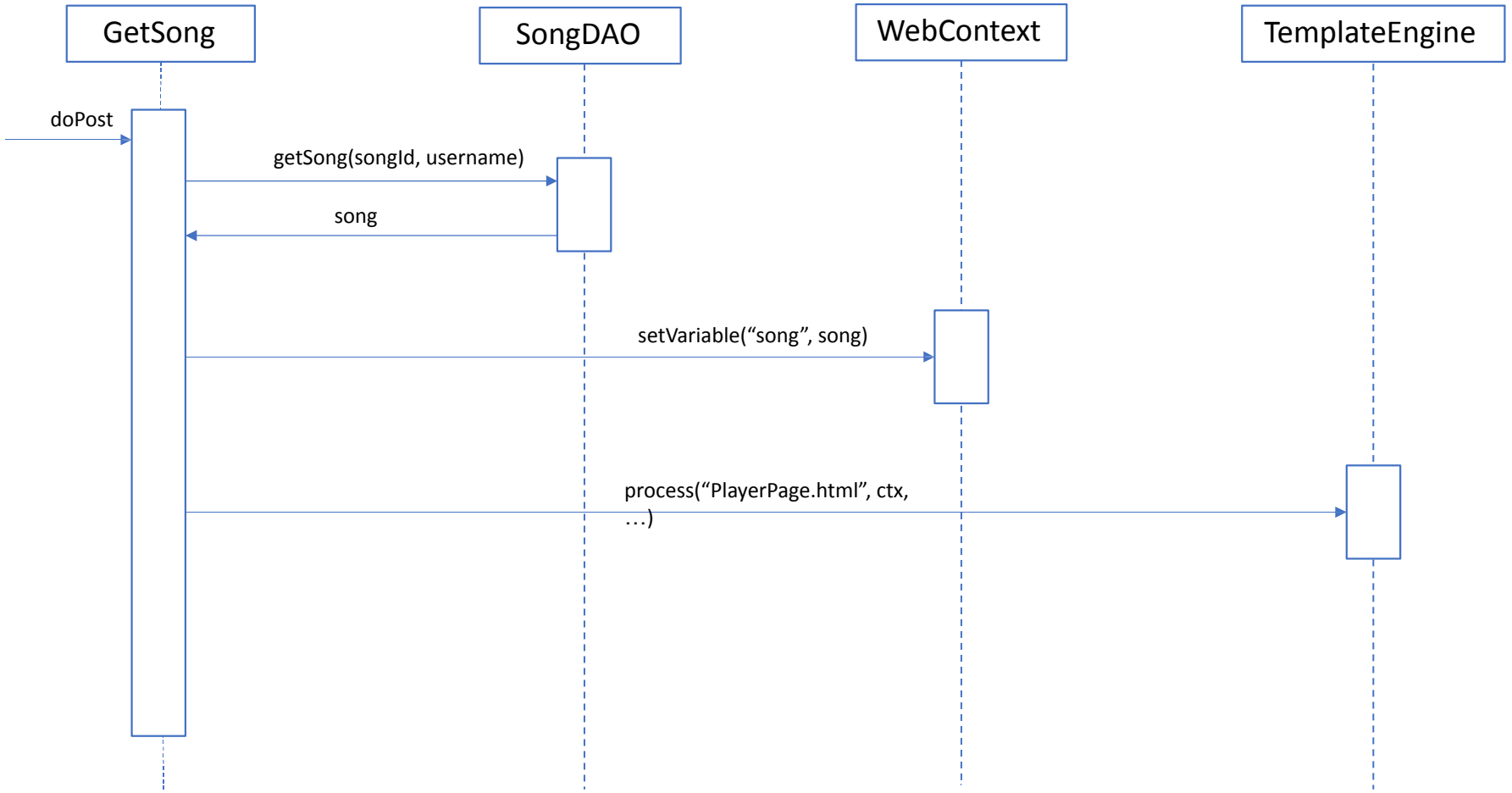
Event: RemoveSongFromPlaylist

POST request from **PlaylistPage.html**
Parameters: songId, playlistId



View: getSong

GET request from **PlaylistPage.html**
Parameters: songId



VERSIONE JavaScript

Specifiche aggiuntive

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.

Cambiamenti alla base di dati:

```
CREATE TABLE PLAYLISTSONG(
        ID_Playlist INT NOT NULL,
      ID_Song INT NOT NULL,
      Username_User VARCHAR(50) NOT NULL,
      Position INT DEFAULT NULL,
      PRIMARY KEY (ID_Playlist,ID_Song),
      FOREIGN KEY (ID_Song) REFERENCES SONG(ID)  ON
      DELETE CASCADE ON UPDATE CASCADE,
      FOREIGN KEY(ID_Playlist) REFERENCES PLAYLIST(ID)  ON
      DELETE CASCADE ON UPDATE CASCADE,
      FOREIGN KEY(Username_User) REFERENCES
      USER(username) ON DELETE CASCADE ON UPDATE
      CASCADE)
};
```

E' stata aggiunta la Colonna "position" per abilitare la funzionalità di ordinamento manuale della playlist.
La stessa base di dati può essere utilizzata per la versione HTML puro, in quanto il dato aggiuntivo non interferisce in alcun modo con il funzionamento dell'applicazione

Componenti

Model objects (JavaBeans)

- User
- Playlist
- Song
- Album

Data Access Objects

- **UserDAO**
  - createUser(username, password,name,surname)
  - findUser(username)
  - checkAuthentication(username,password)
  - getAllUserSongs(username, getAlbumCovers)
  - findPlaylists(username)
  - getAllAlbums(username)
  - deleteAccount(username)
- **PlaylistDAO**
  - createPlaylist(title, songList, username)
  - findPlaylist(username)
  - checkPlaylistOwner(playlistID, username)
  - getPlaylistTitle(playlistID, username)
  - countSongs(playlistID, username)
  - getSongsNotInPlaylist(username, playlistID)
  - getSongsFiltered(playlistID, username, offset)
  - getSongs(playlistID, username)
  - removeSongFromPlaylist(songID, playlistID, username)
  - deletePlaylist(playlistID, username)
  - **SetPlaylistOrder**(playlistID, songsID, username)
- **SongDAO**
  - uploadSongExistingAlbum(songTitle, albumTitle,singer,publicationYear,genre,imagePath,songFilePath, usernameUser)
  - existAlbum(albumTitle, username)
  - addSongToPlaylist(songID, playlistID, username)
  - addSongsToPlaylist(songList, playlistID,username)
  - getSong(songID, username)
  - checkSongOwner(songID, username)
  - deleteSong(songID, username)

Componenti

Controllers(servlets)

- **Authentication**
  - CheckLogin
  - Logout
  - CheckRegistration
  - DeleteAccount
- **Get content**
  - GetAllPlaylist
  - GetAllSong
  - GoToHomePage
- **Create content**
  - AddSongsToPlalylist
  - CreatePlaylist
  - UploadSong
  - ReorderPlaylistSong
- **Delete content**
  - DeletePlaylist
  - DeleteSong
  - RemoveSongFromPlaylist

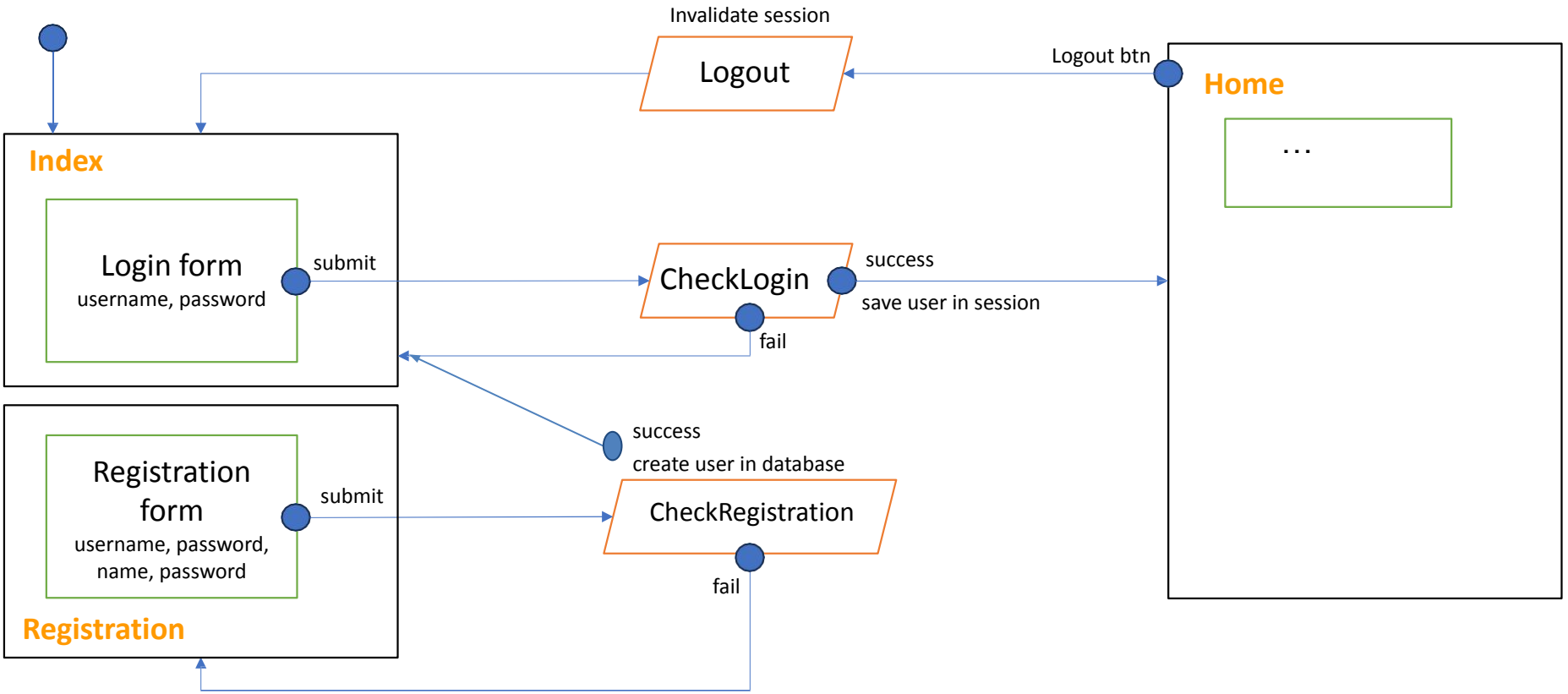
Views(Templates)

- Index(Login)
- Registration
- HomePage

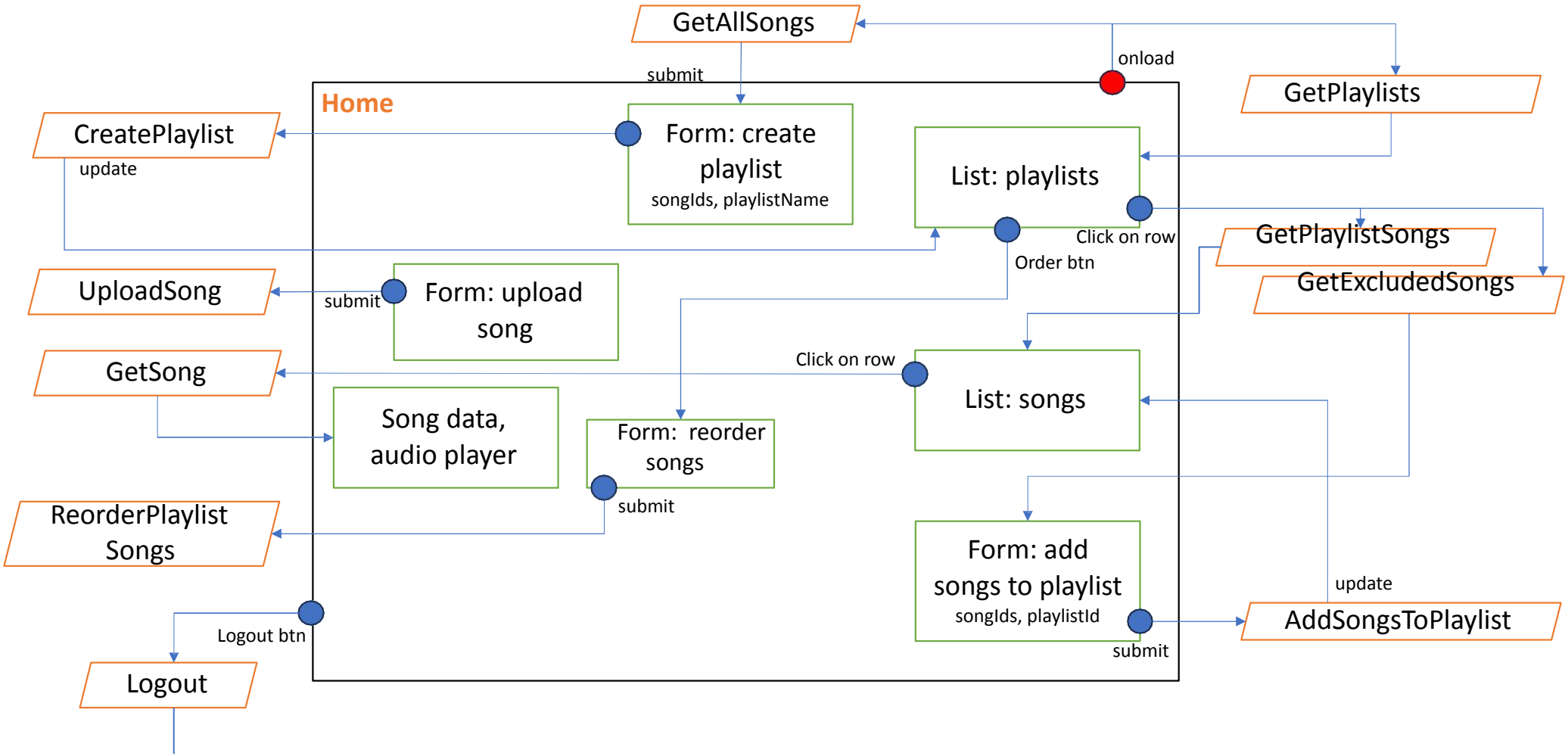
Utilities

- ConnectionHandler

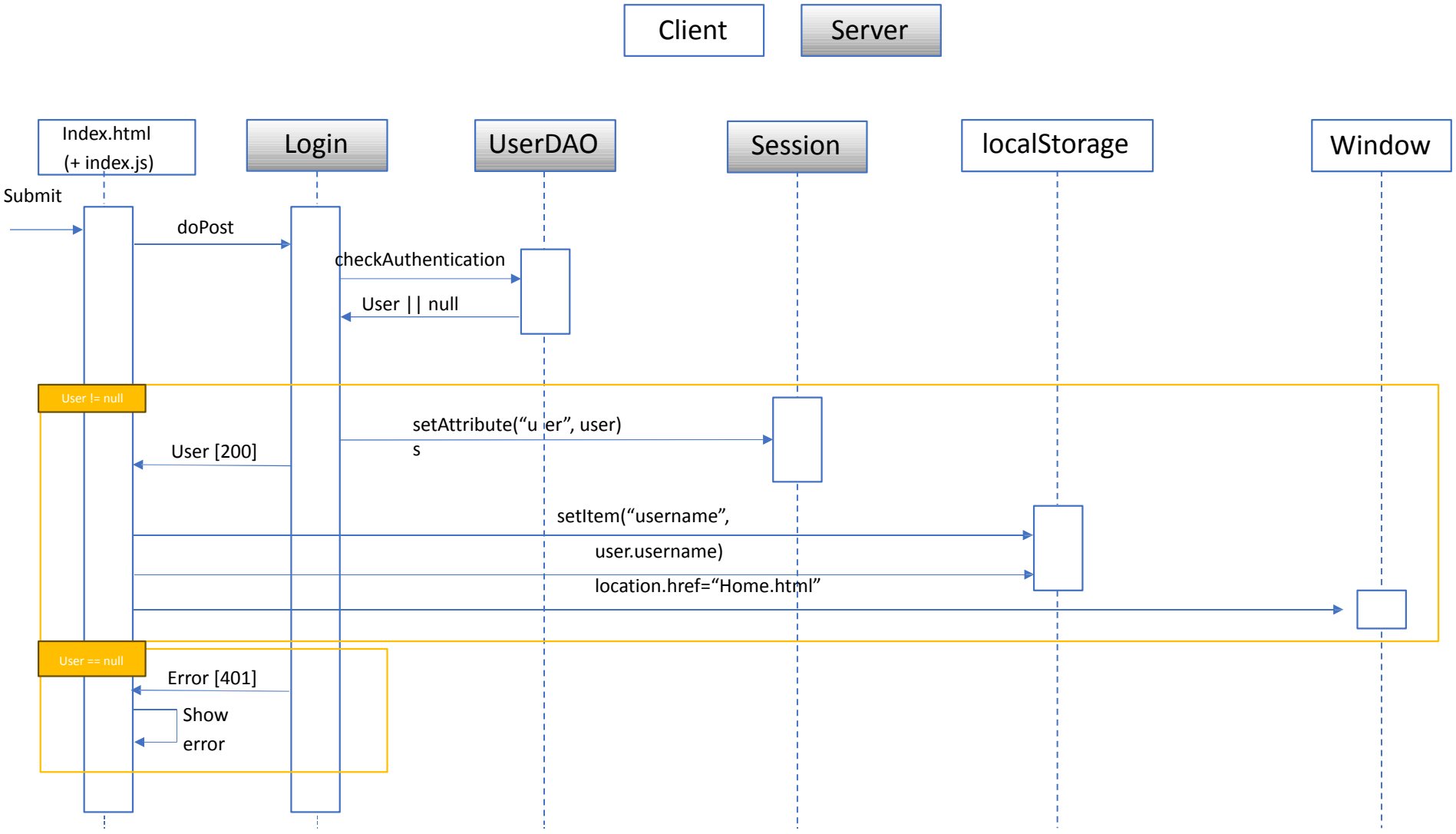
Application design (login/create account)



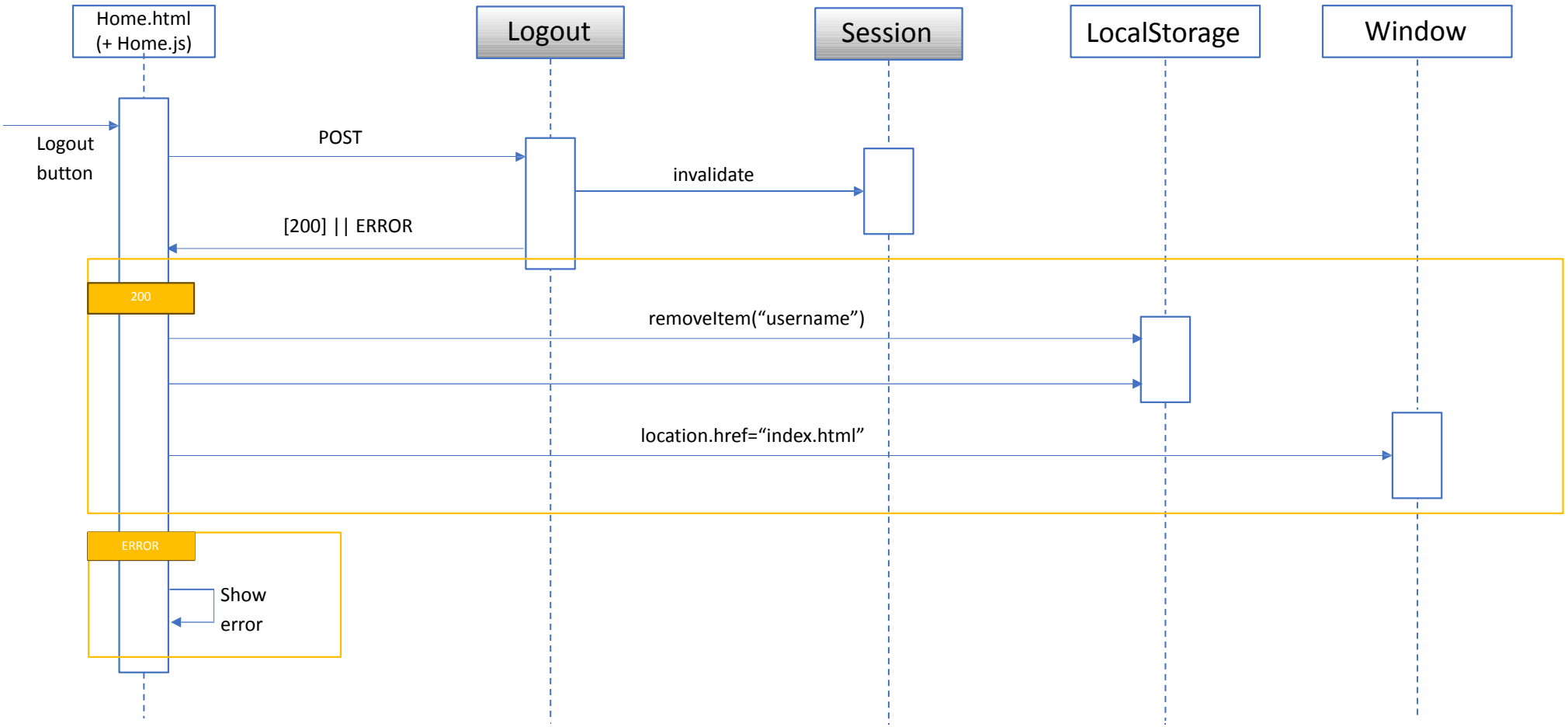
Application design



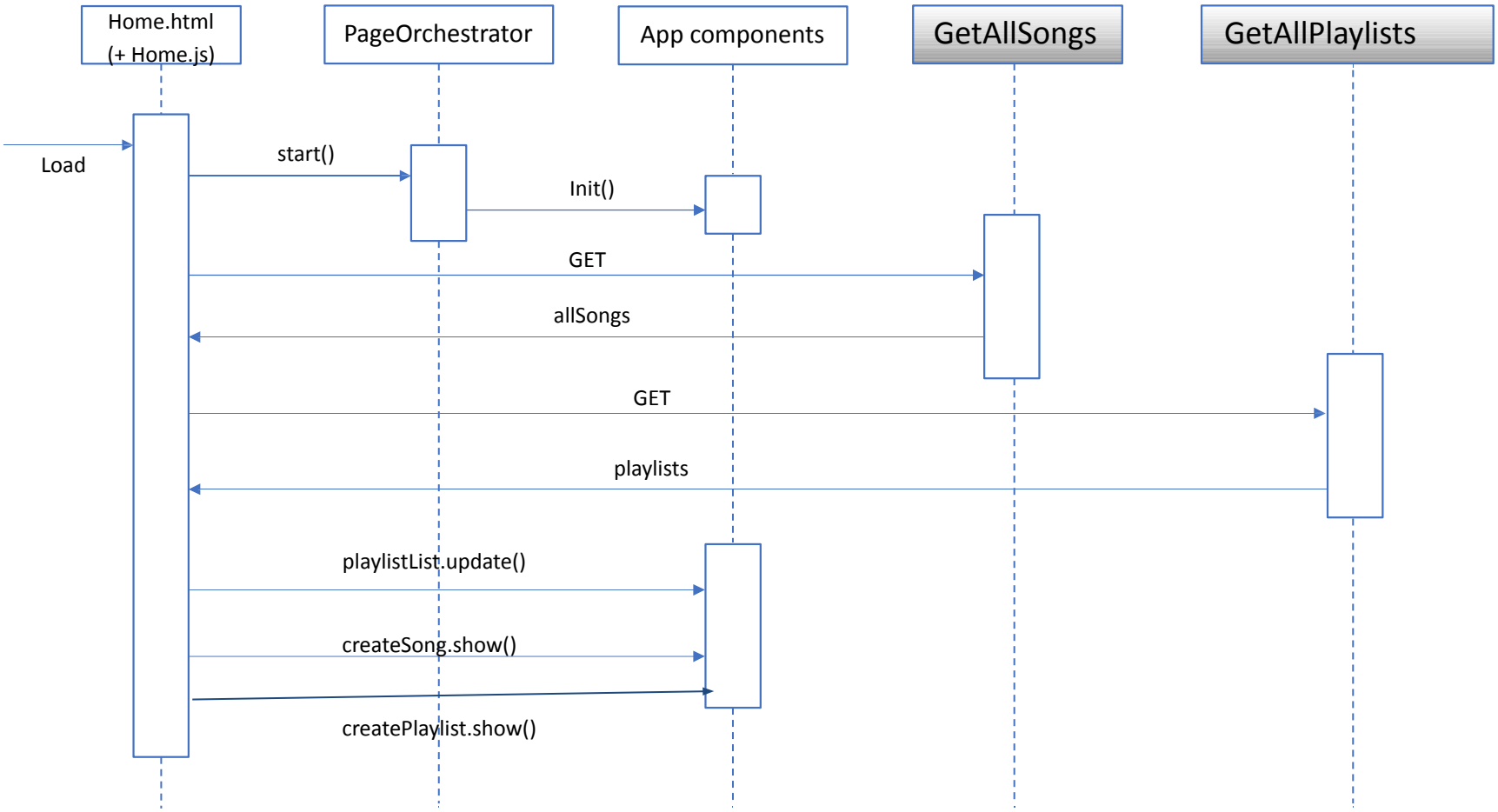
Event: Login


Client
Server
Index.html
(+ index.js)
Login
UserDAO
Session
localStorage
Window
Submit
doPost
checkAuthentication
User || null
User != null
setAttribute(“u er”, user)
s
User [200]
setItem(“username”,
user.username)
location.href=“Home.html”
User == null
Error [401]
Show
error

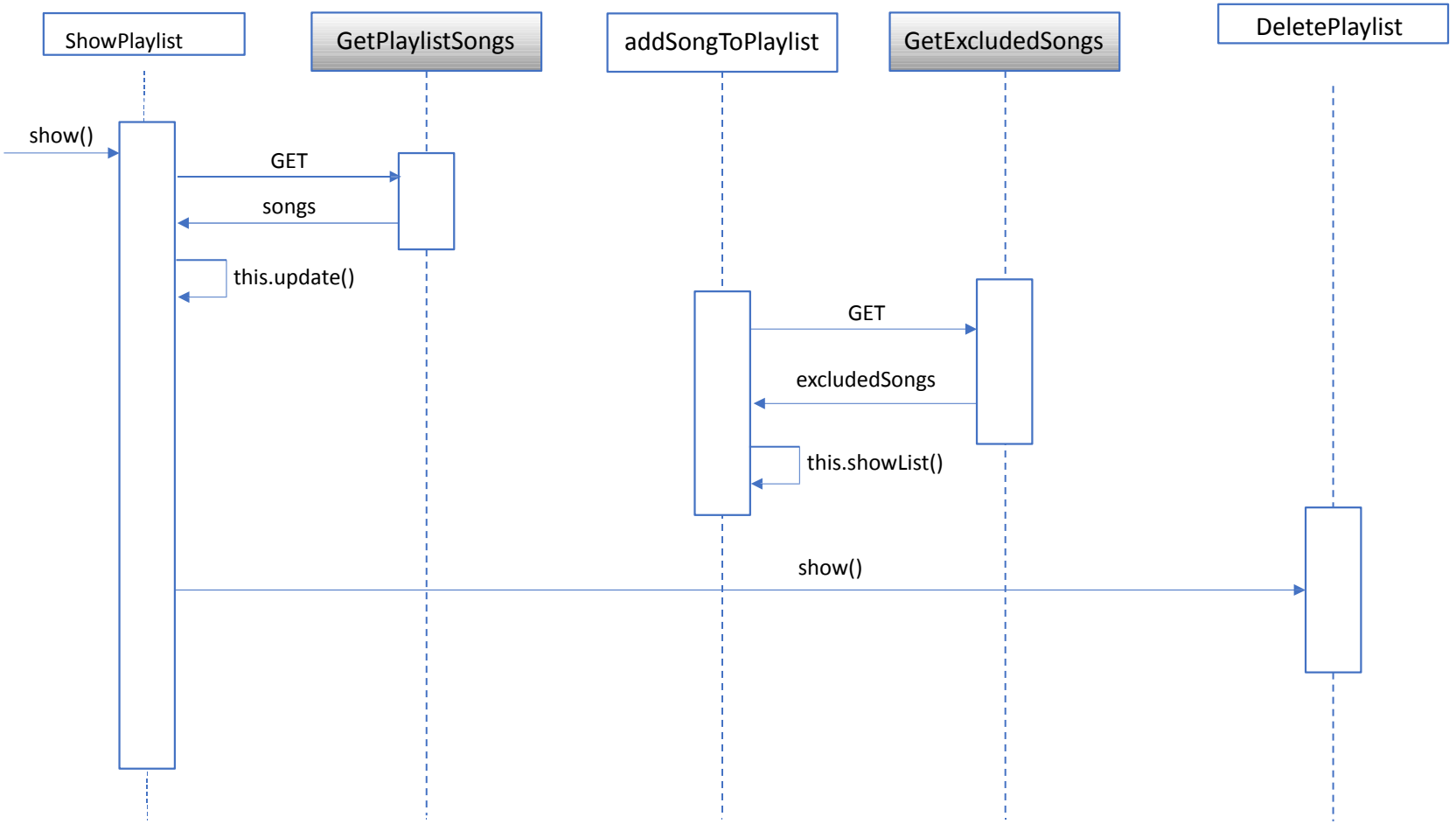
Event: Logout



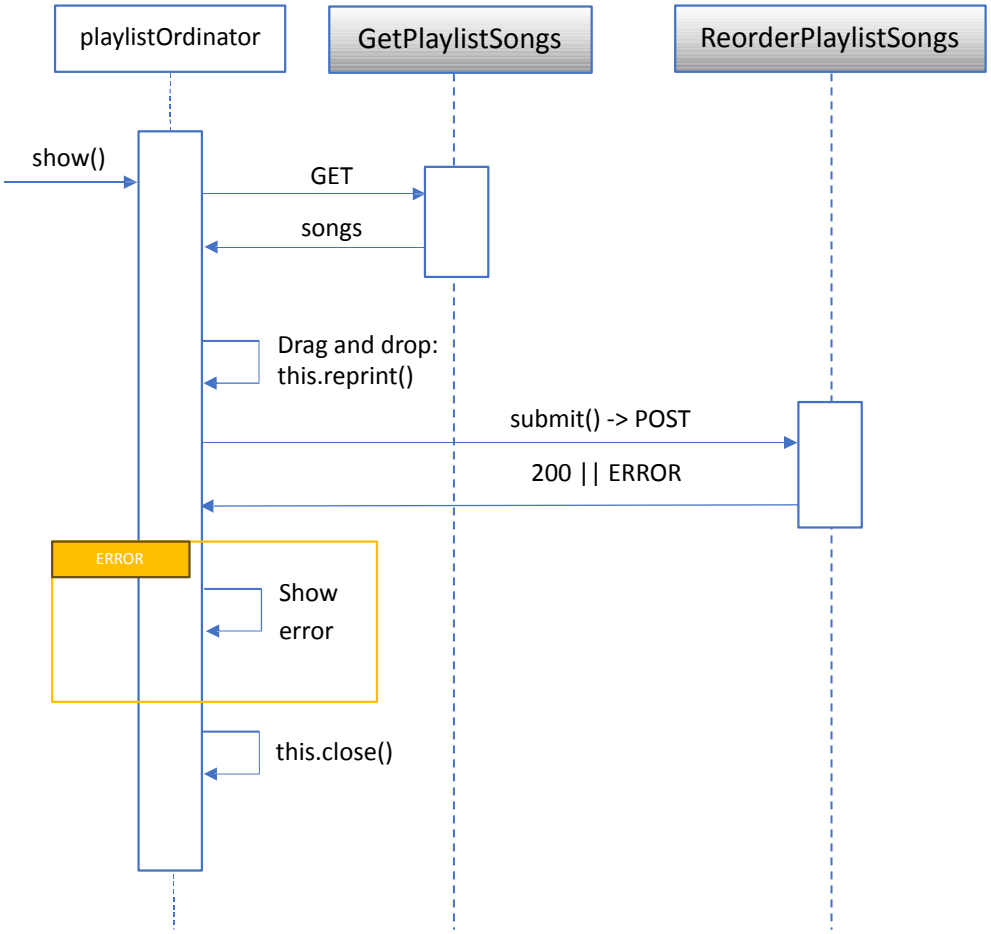
View: Application - onload



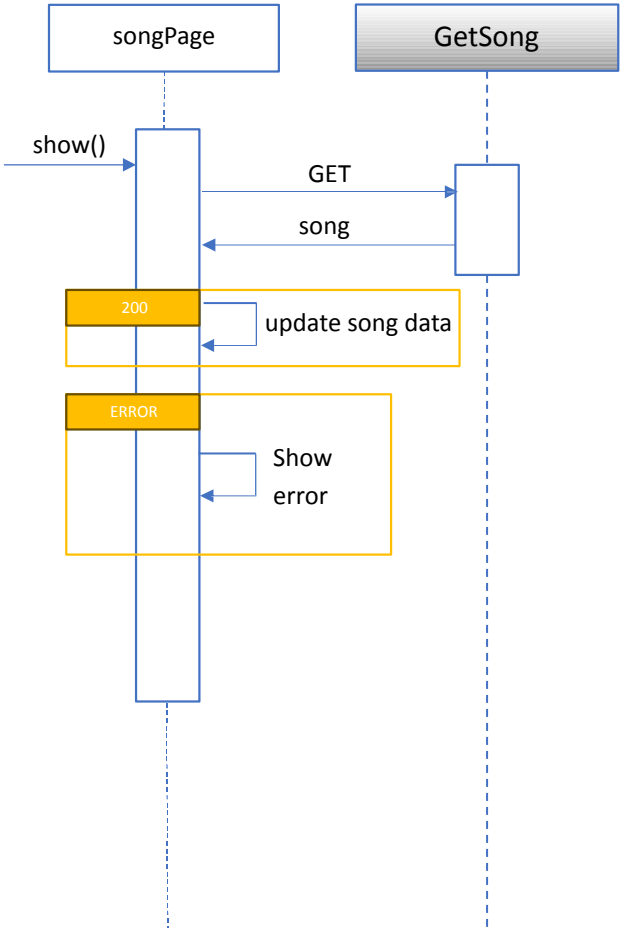
Event: click on playlist



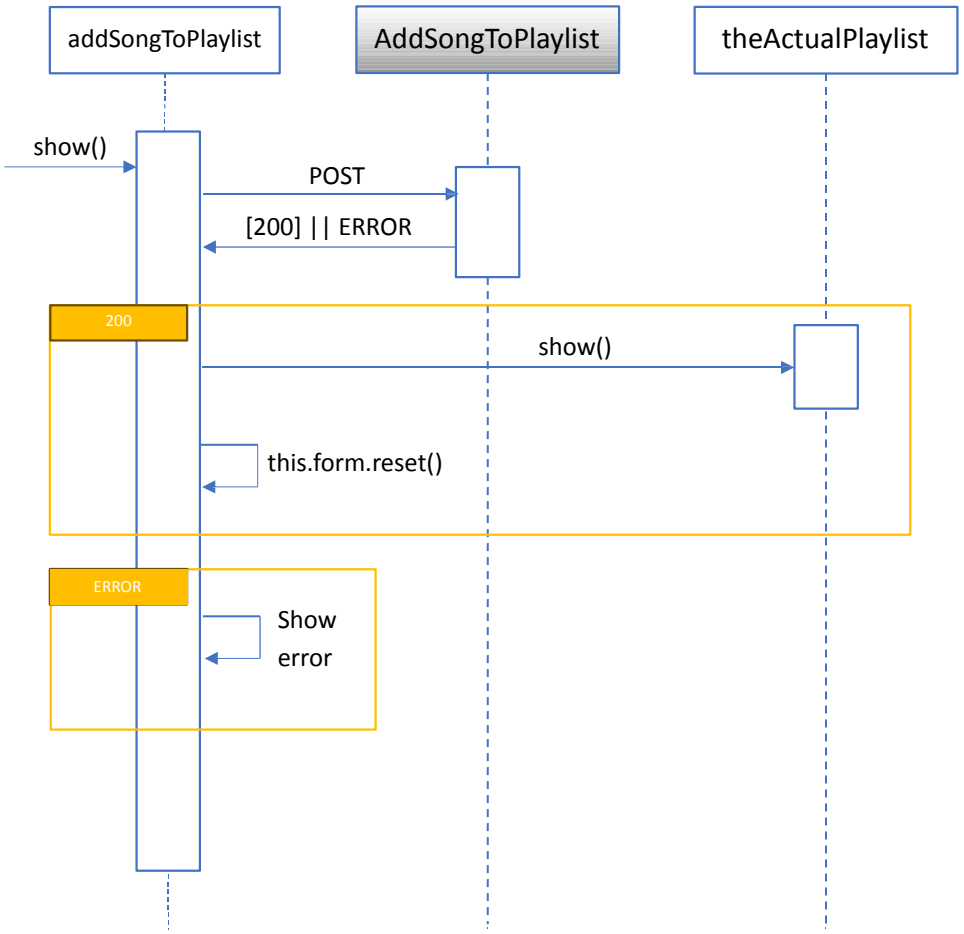
Event: reorder playlist



Event: click on song



Event: add song to playlist



Event: upload song