

# **THE PLAYLIST**

## **TIW PROJECT 2022/2023**

**Francesco Rita & Mattia Campana**

# TASK

Un'applicazione web consente la gestione di una playlist di brani musicali. Playlist e brani sono personali di ogni utente e non condivisi. 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. 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 ordinati per data decrescente dall'anno di pubblicazione dell'album. 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, una form per caricare un brano con tutti i dati relativi e una form per creare una nuova playlist. La creazione di una nuova playlist richiede di 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 data decrescente dell'album di pubblicazione. 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 una form che consente di selezionare e aggiungere un brano alla playlist corrente, se non già presente nella 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.

# **HTML**

# SCHEMA ANALYSIS



Un'applicazione web consente la gestione di una playlist di brani musicali. Playlist e brani sono personali di ogni utente e non condivisi. 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.

...

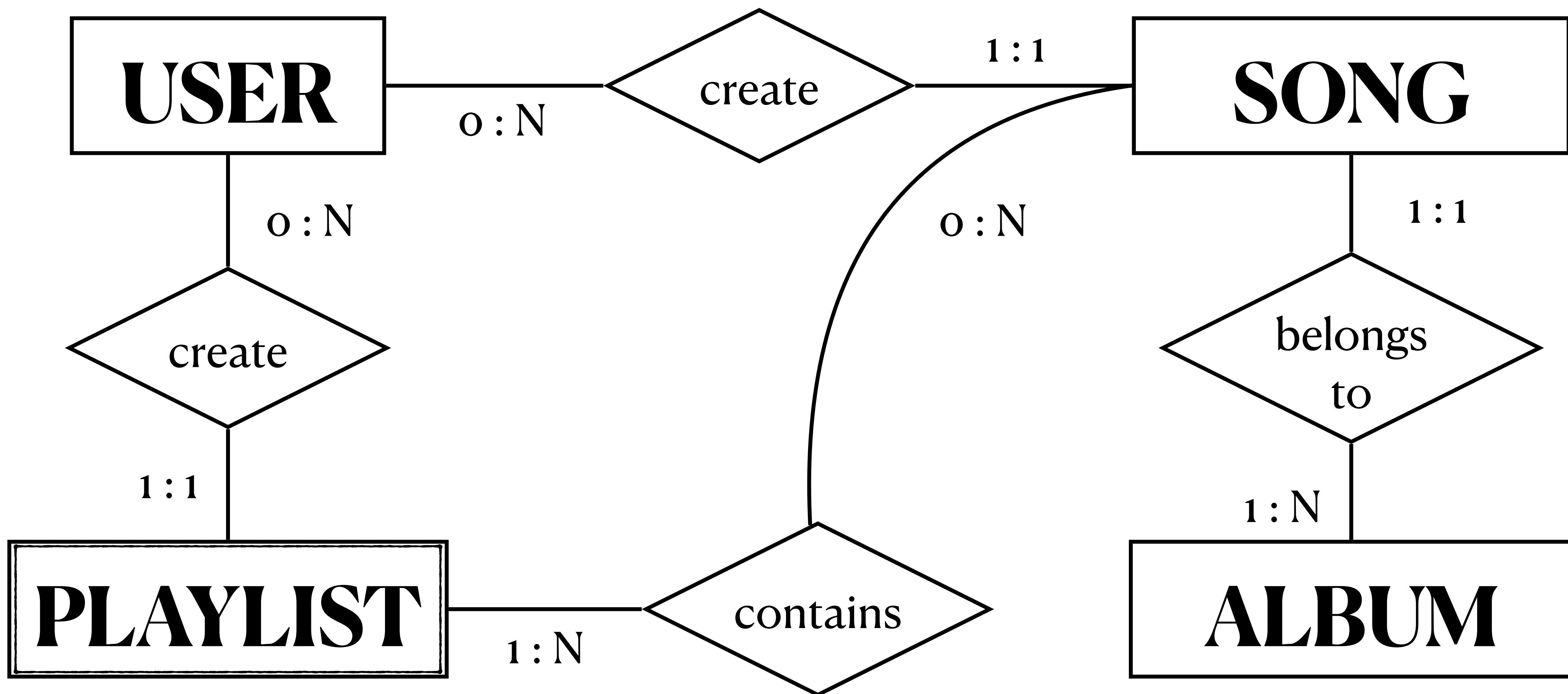
La creazione di una nuova playlist richiede di selezionare uno o più brani da includere.

...

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.

ENTITIES - ATTRIBUTES - RELATIONSHIP

# DATABASE DESIGN



# LOCAL DATABASE SCHEMA

- create table USER(  
UserName varchar(50) primary key,  
Password varchar(50) not null  
);
- create table ALBUM(  
Id int primary key auto\_increment,  
Title varchar(50) not null,  
FileImage varchar(50) not null,  
Singer varchar(50) not null  
PublicationYear year not null  
);
- create table SONG(  
Id int primary key auto\_increment,  
Title varchar(50) not null,  
Genre varchar(50) not null,  
FileAudio varchar(50) not null,  
User varchar(50) not null  
references USER(UserName) on  
delete cascade on update cascade,  
Album int not null references  
ALBUM(Id) on delete cascade on  
update cascade  
);

# LOCAL DATABASE SCHEMA

- create table PLAYLIST(  
    Name varchar(50) not null,  
    UserName varchar(50) not null reference USER(UserName) on delete cascade on update cascade,  
    CreationDate date not null,  
    primary key (Name, UserName)  
);
- create table Contains(  
    PlaylistName varchar(50) not null,  
    PlaylistUser varchar(50) not null,  
    Song int not null references SONG(Id) on delete cascade on update cascade,  
    primary key(PlaylistName, PlaylistUser, Song),  
    foreign key (PlaylistName, PlaylistUser) references PLAYLIST(Name, UserName) on delete cascade  
    on update cascade  
);

# APPLICATION REQUIREMENT ANALYSIS

A seguito del login, l'utente accede all'HOME PAGE che presenta l'elenco delle proprie playlist, ordinate per data di creazione decrescente, una form per caricare un brano con tutti i dati relativi e una form per creare una nuova playlist.

La creazione di una nuova playlist richiede di 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 data decrescente dell'album di pubblicazione. 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 una form che consente di selezionare e aggiungere un brano alla playlist corrente, se non già presente nella 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.

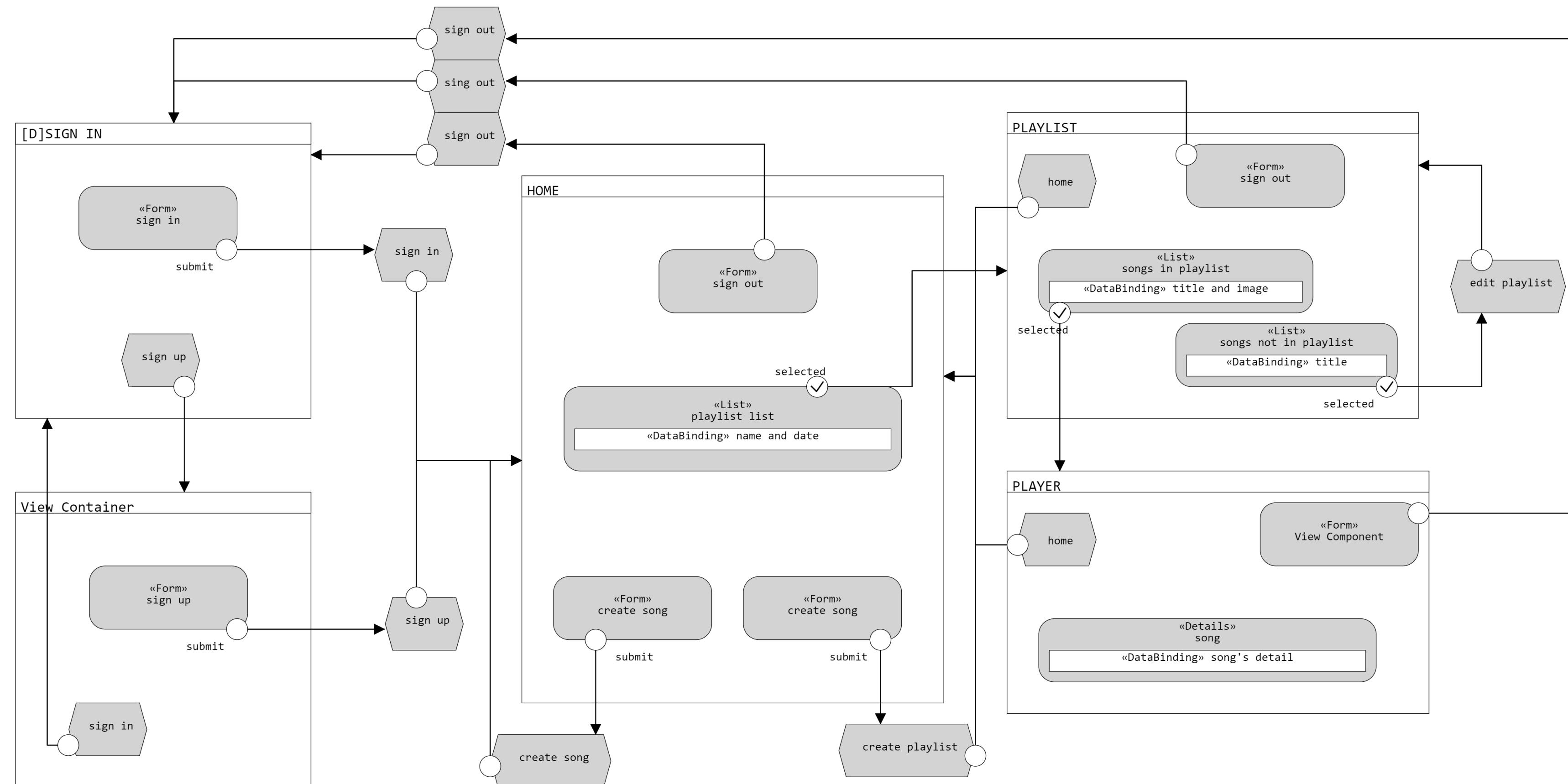
PAGES

EVENTS

ACTIONS

VIEW COMPONENTS

# APPLICATION DESIGN



# VIEWS AND COMPONENTS

MODEL OBJECTS (BEANS)

DATA ACCESS OBJECTS (DAO)

CONTROLLERS (SERVLET)

FILTERS AND UTILS

VIEWS (TEMPLATES)

# MODEL OBJECTS

## BEANS

~ SONG

- ID, TITLE, GENRE, FILEAUDIO, ALBUM

~ ALBUM

- ID, TITLE, FILEIMAGE, SINGER, YEAR

~ PLAYLIST

- NAME, CREATIONDATE

# DATA ACCESS OBJECTS

## DAO

- PlaylistDAO:
  - AddPlaylist
  - SongAlreadyIn
  - + AddSongToPlaylist
  - + AddPlaylistWithSongs
  - + BelongTo
  - + AllPlaylist
  - + Taken
- UserDAO:
  - Taken
  - + Authentication
  - + Registration

# DATA ACCESS OBJECTS

## DAO

- SongDAO:
  - FindAlbumId
  - AddAlbum
  - SongAlreadyIn
  - addSong
  - + AddSongAndAlbum
  - + GetSongTitleAndImage
  - + GetSongNotInPlaylist
  - + GetSongByUser
  - + PlaySong
  - + BelongTo
  - + GetNumOfSongByUser
  - + ImgBelongTo
  - + AudioBelongTo

# CONTROLLER SERVLET

- CreatePlaylistServlet
- CreateSongServlet
- EditPlaylistServlet
- GetImageServlet
- GetSongServlet
- HomeServlet
- PlayerServlet
- PlaylistServlet
- SignInServlet
- SignUpServlet
- SignOutServlet

# VIEWS

## TEMPLATE



welcome page

HOME

PLAYLIST

PLAYER

# FILTERS AND UTILS

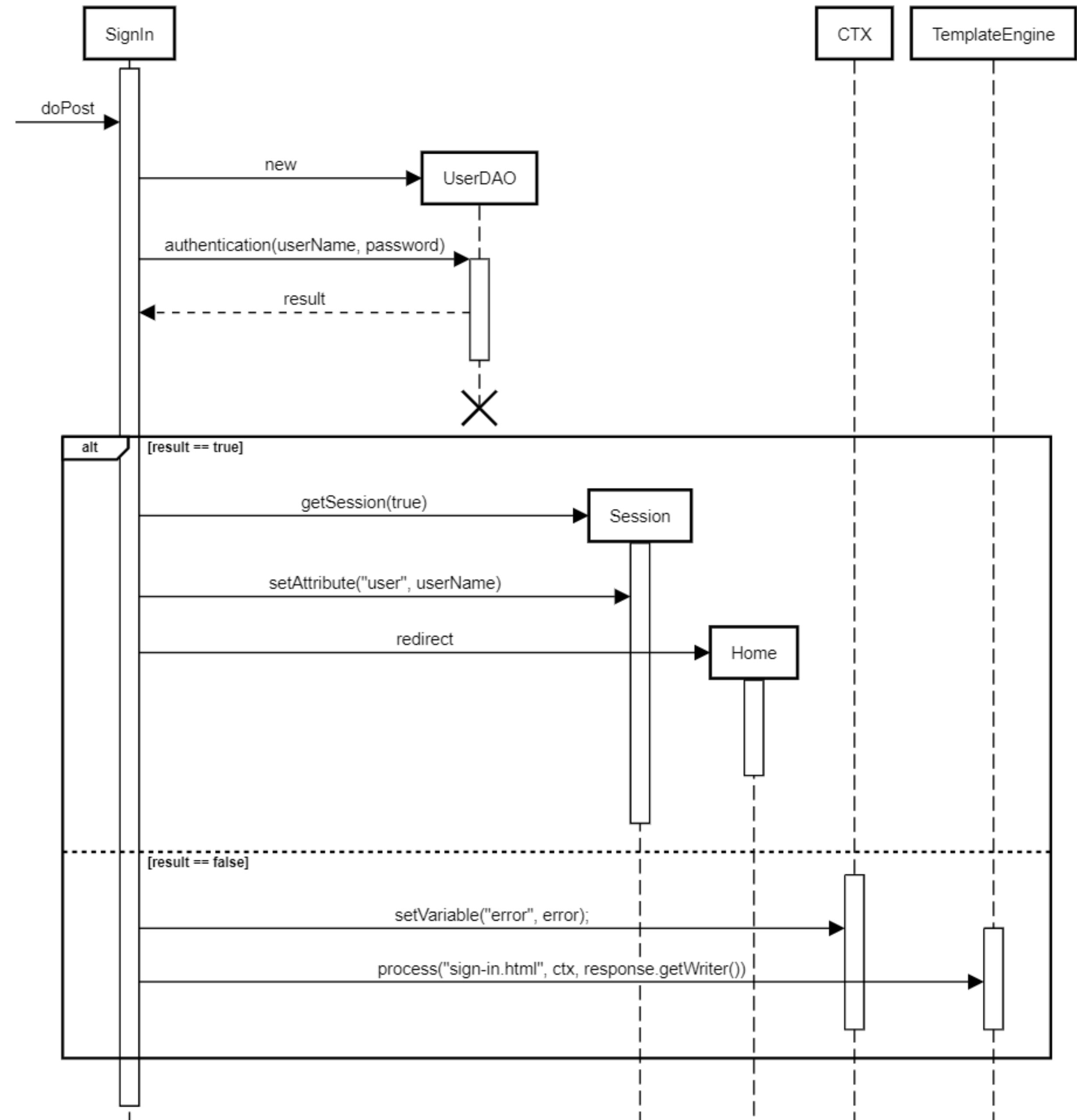
- LoggedChecker
- NotLoggedChecker
- PlayerChecker
- PlaylistChecker
- ConnectionHandler
- TemplateHandler

# SEQUENCE DIAGRAM

**every DAO constructor has the database connection as parameter**

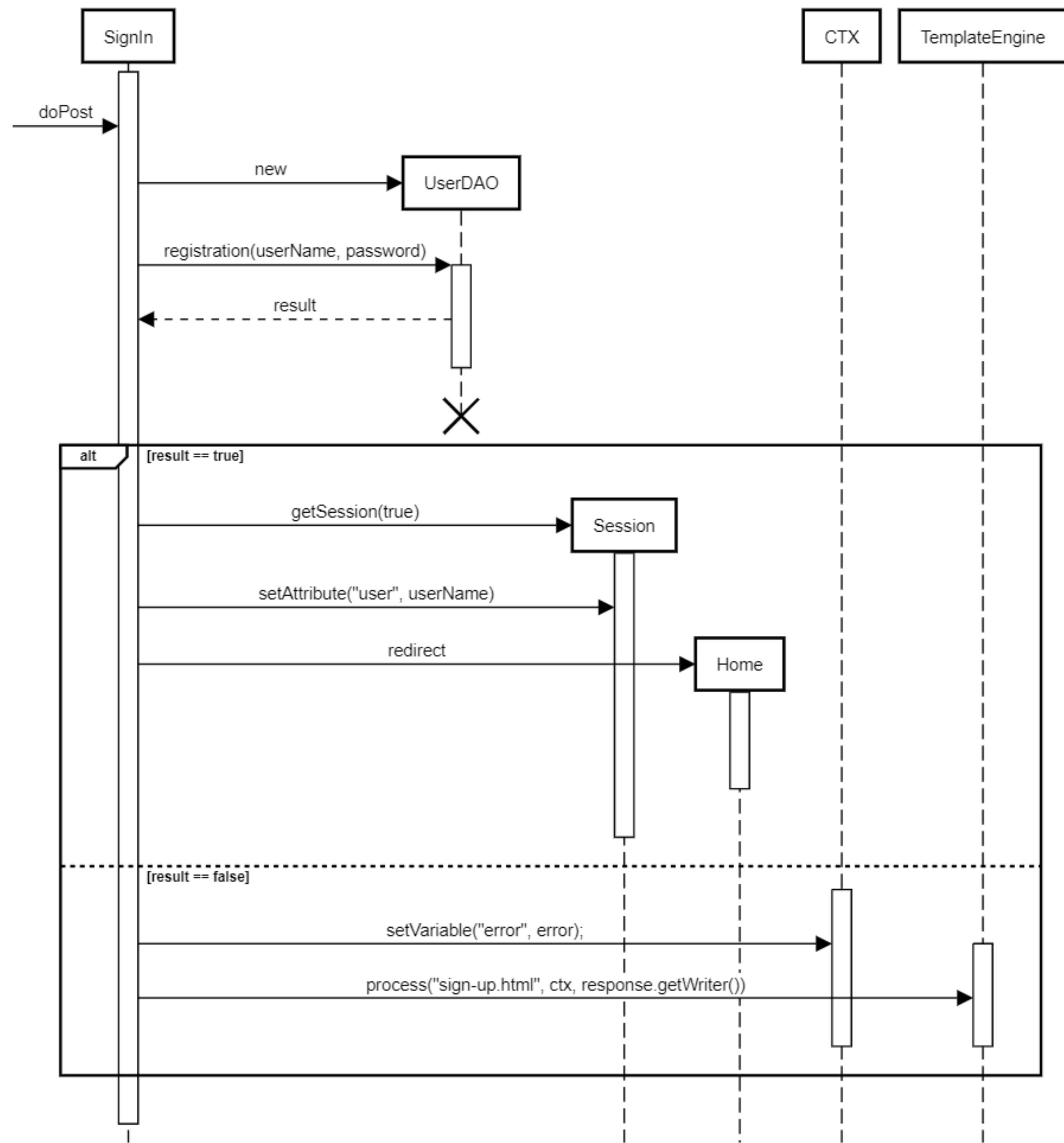
# Sign In

SIGN IN

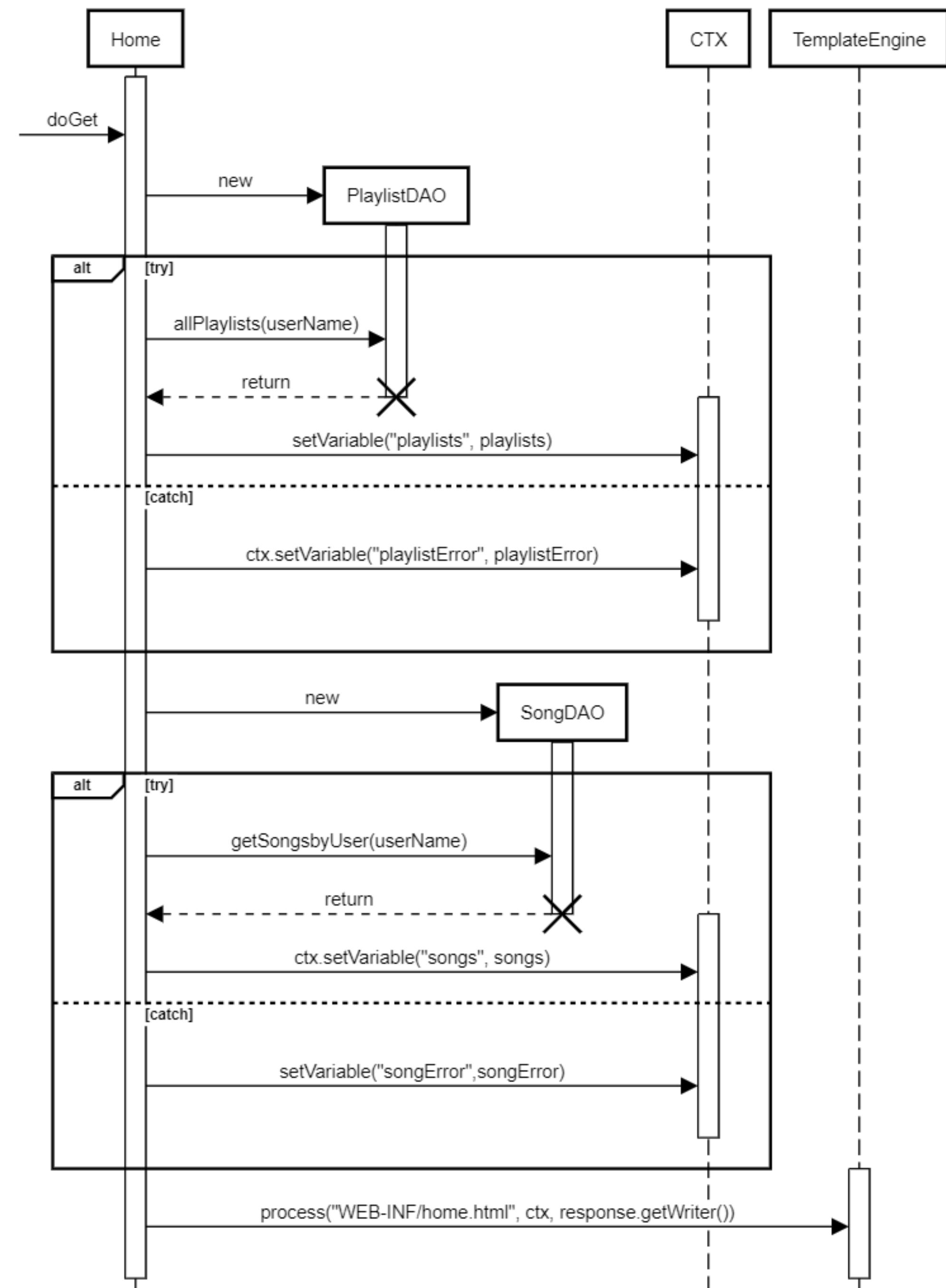


# SignUp

SIGN UP

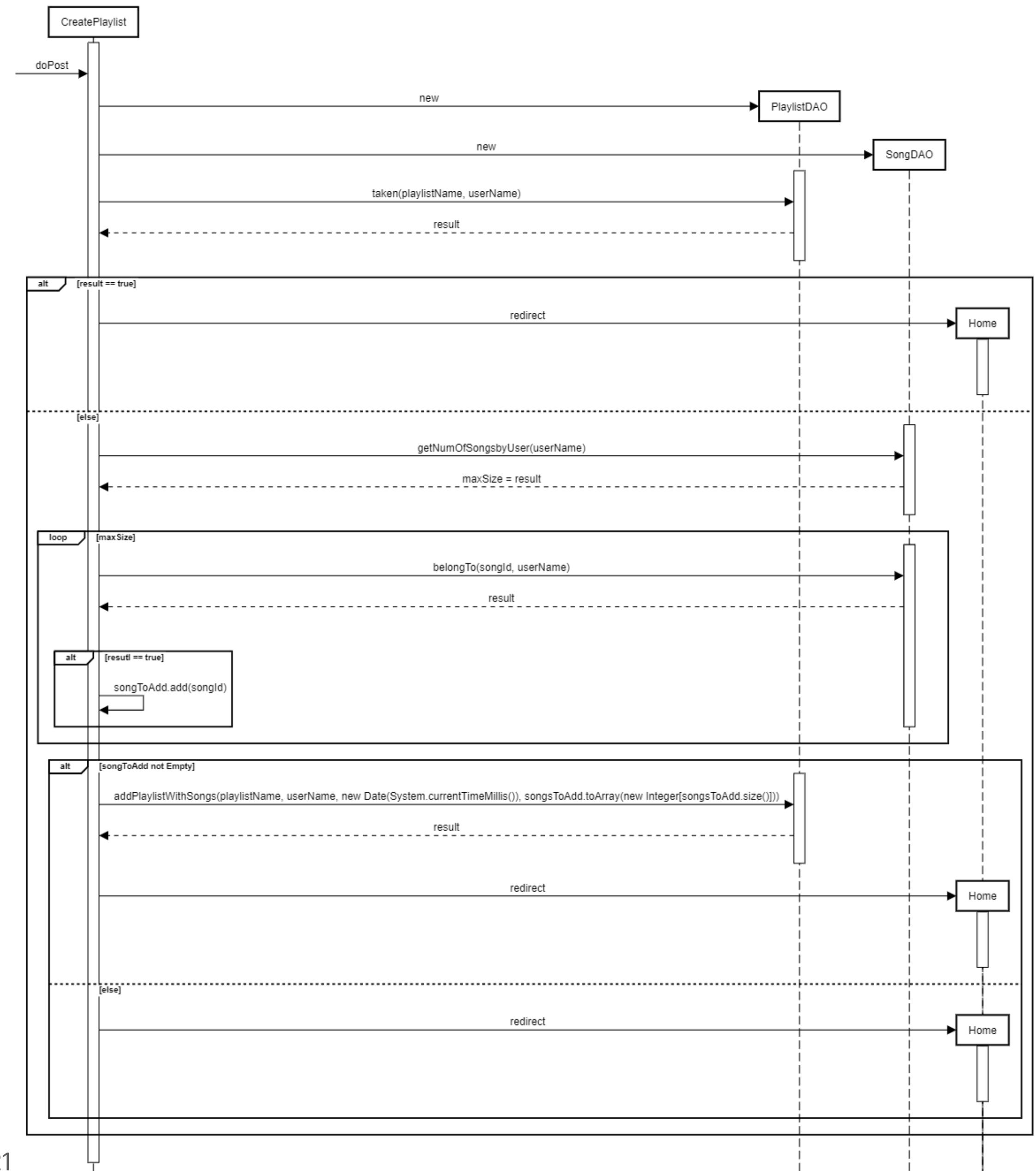


# Home Page

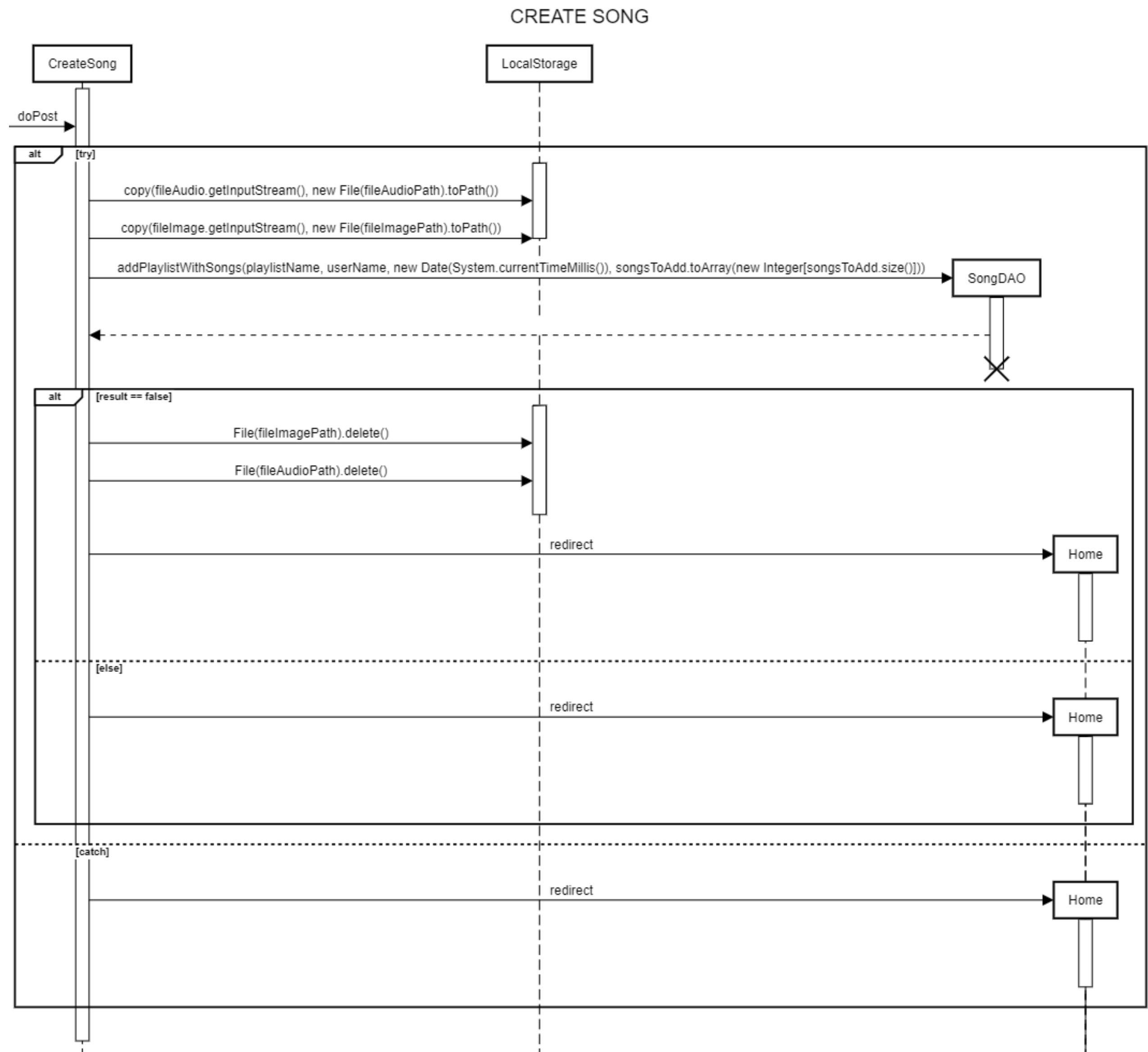


# Create Playlist

CREATE PLAYLIST

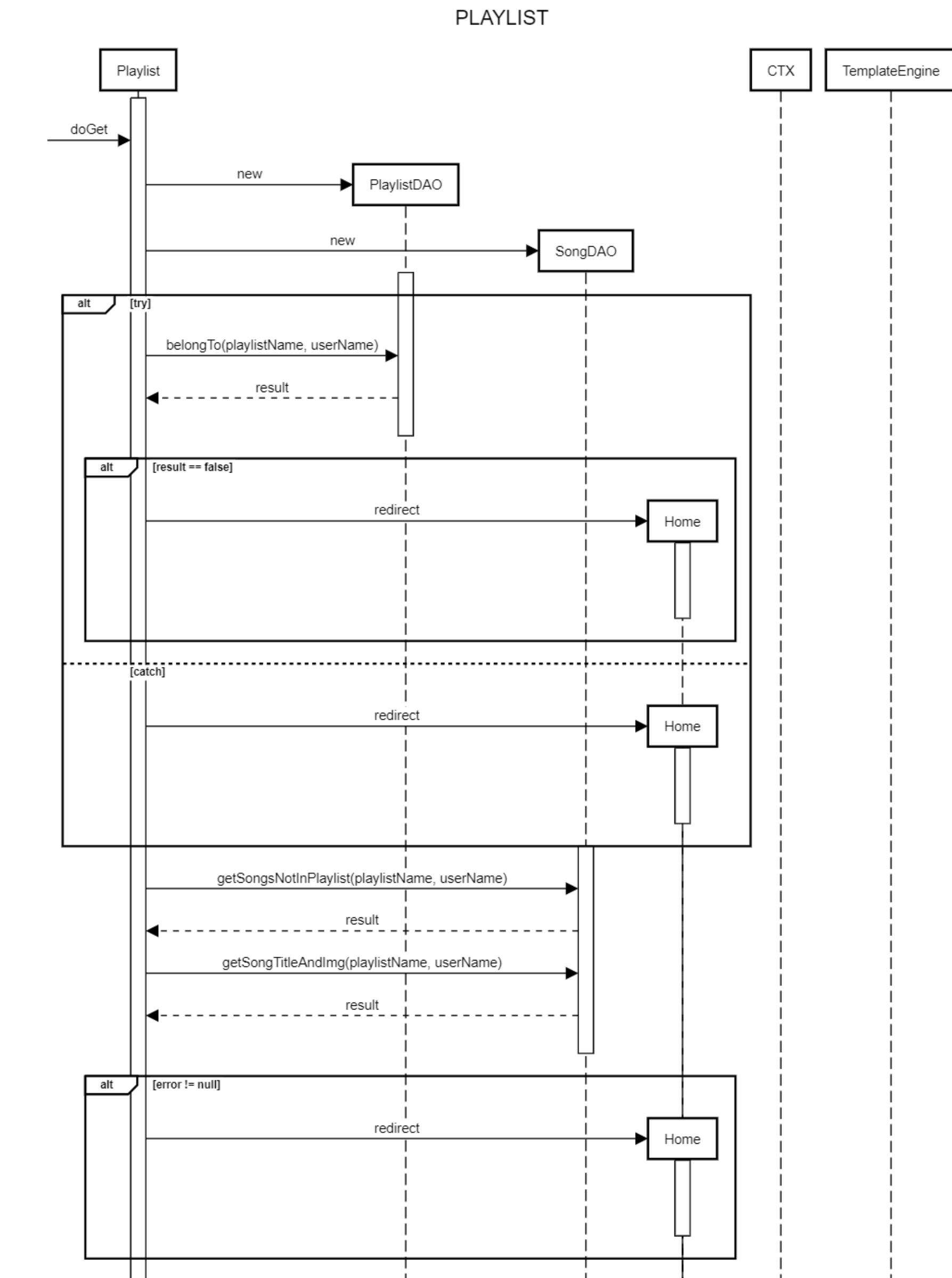


# Create Song

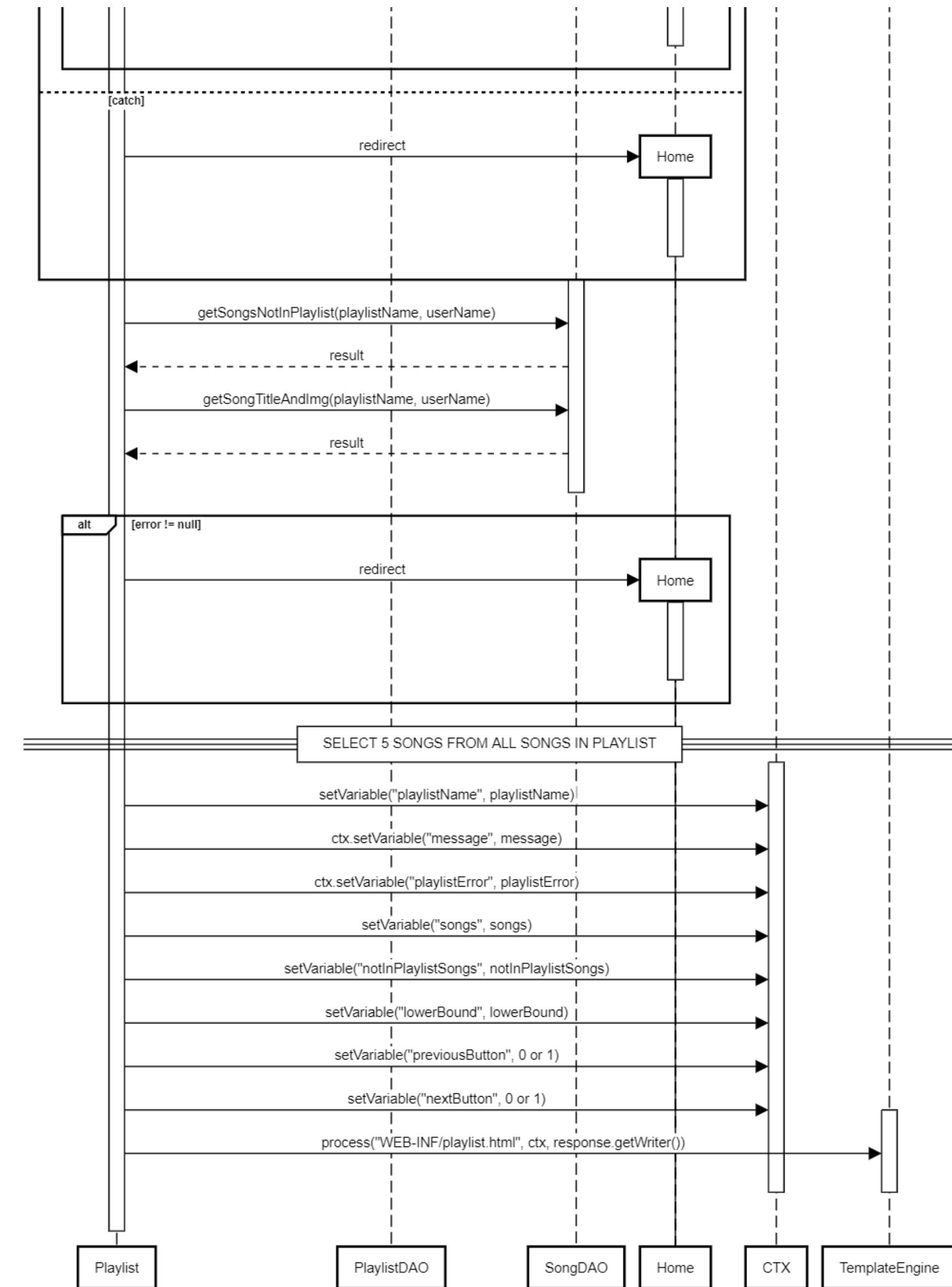


# Playlist Page

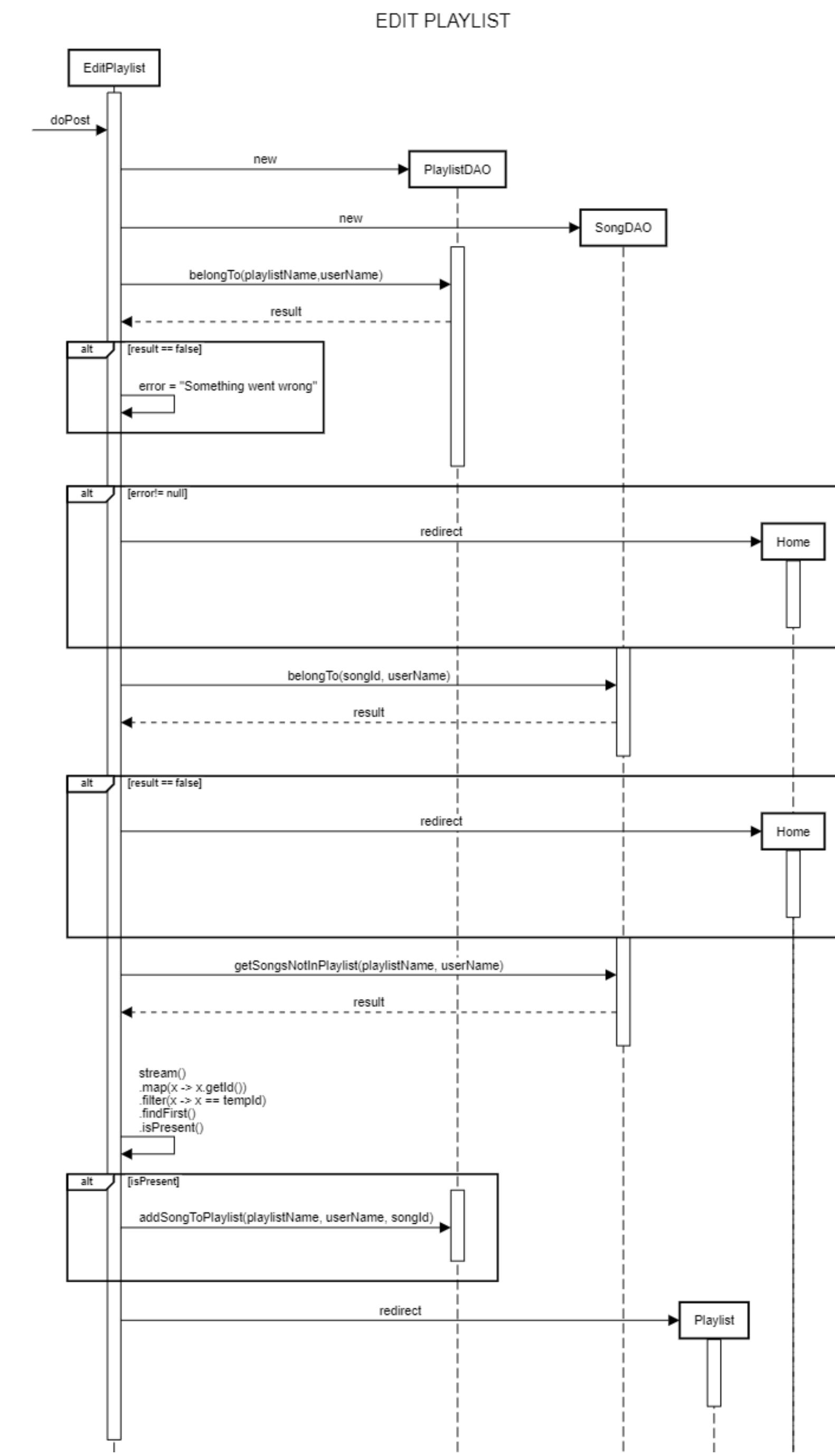
GO NEXT PAGE



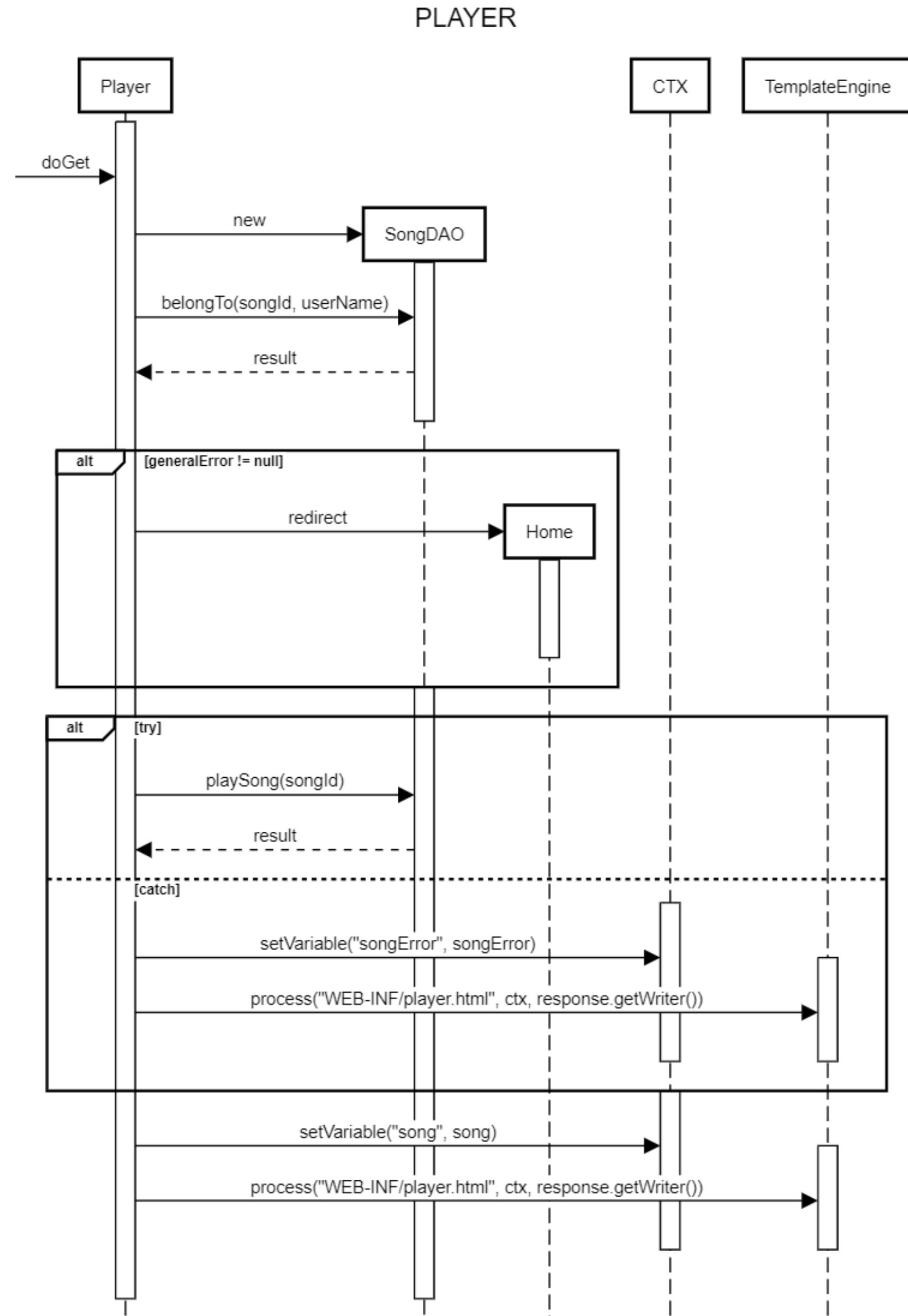
# Playlist Page



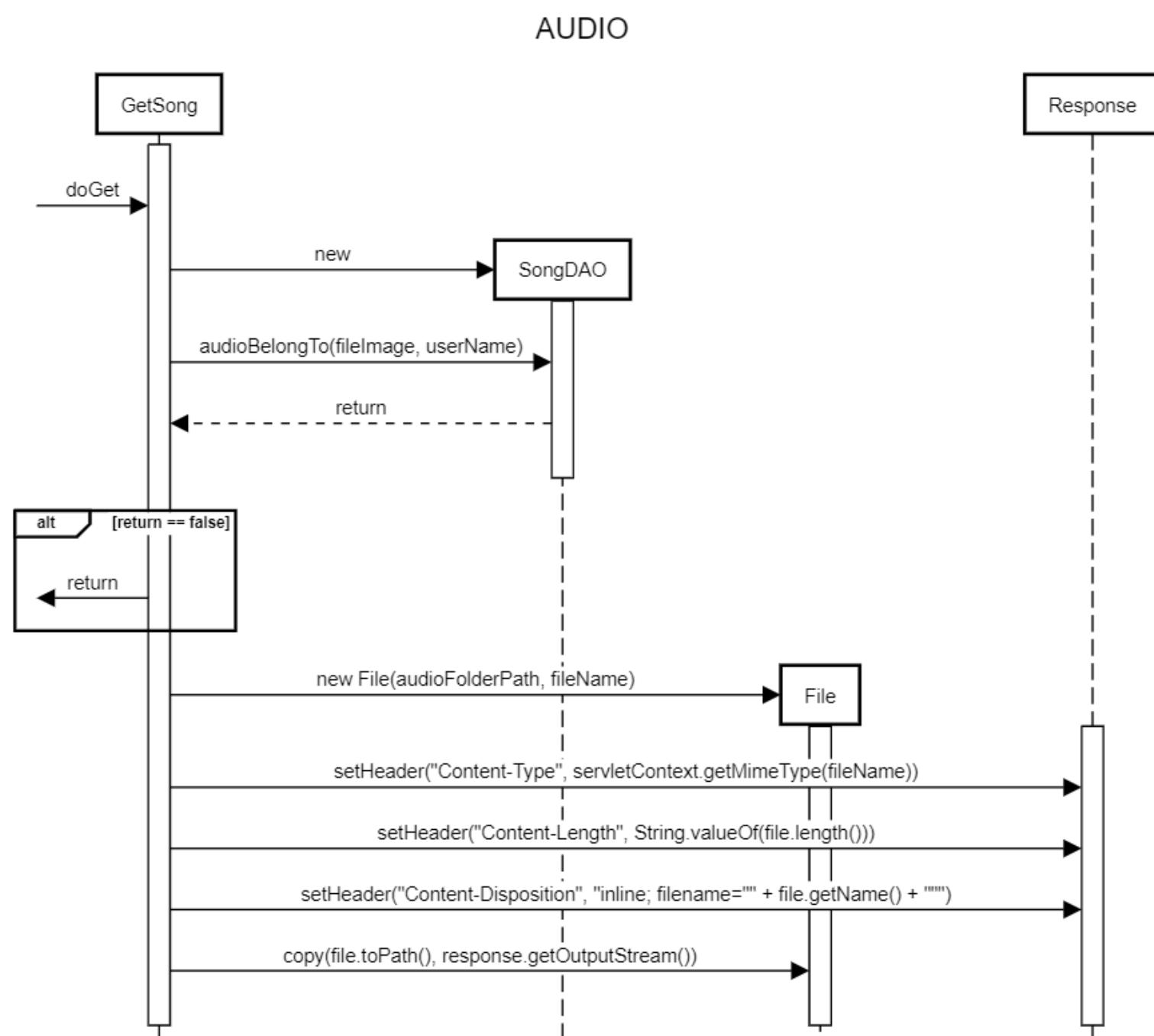
# Edit Playlist



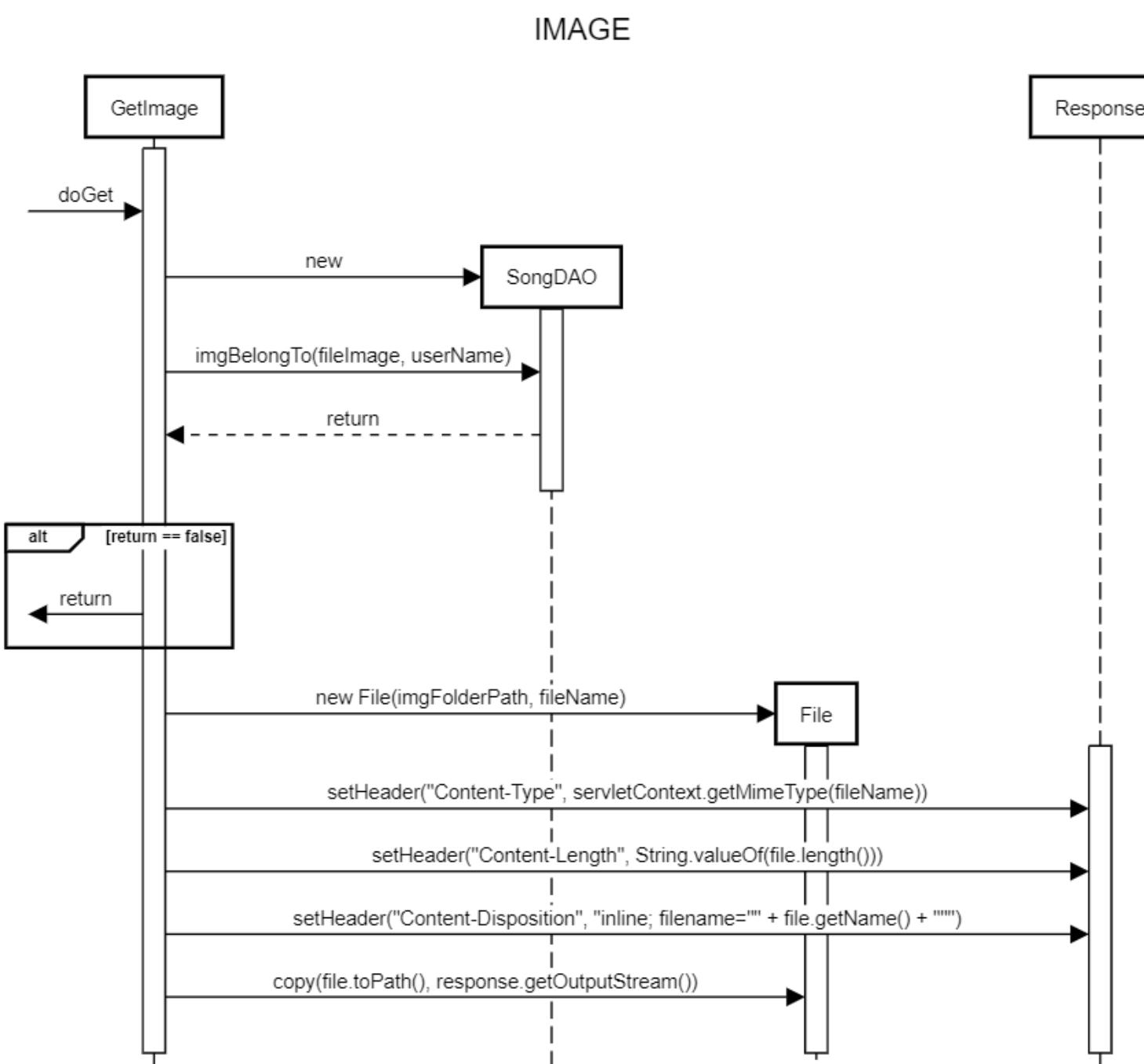
# Player Page



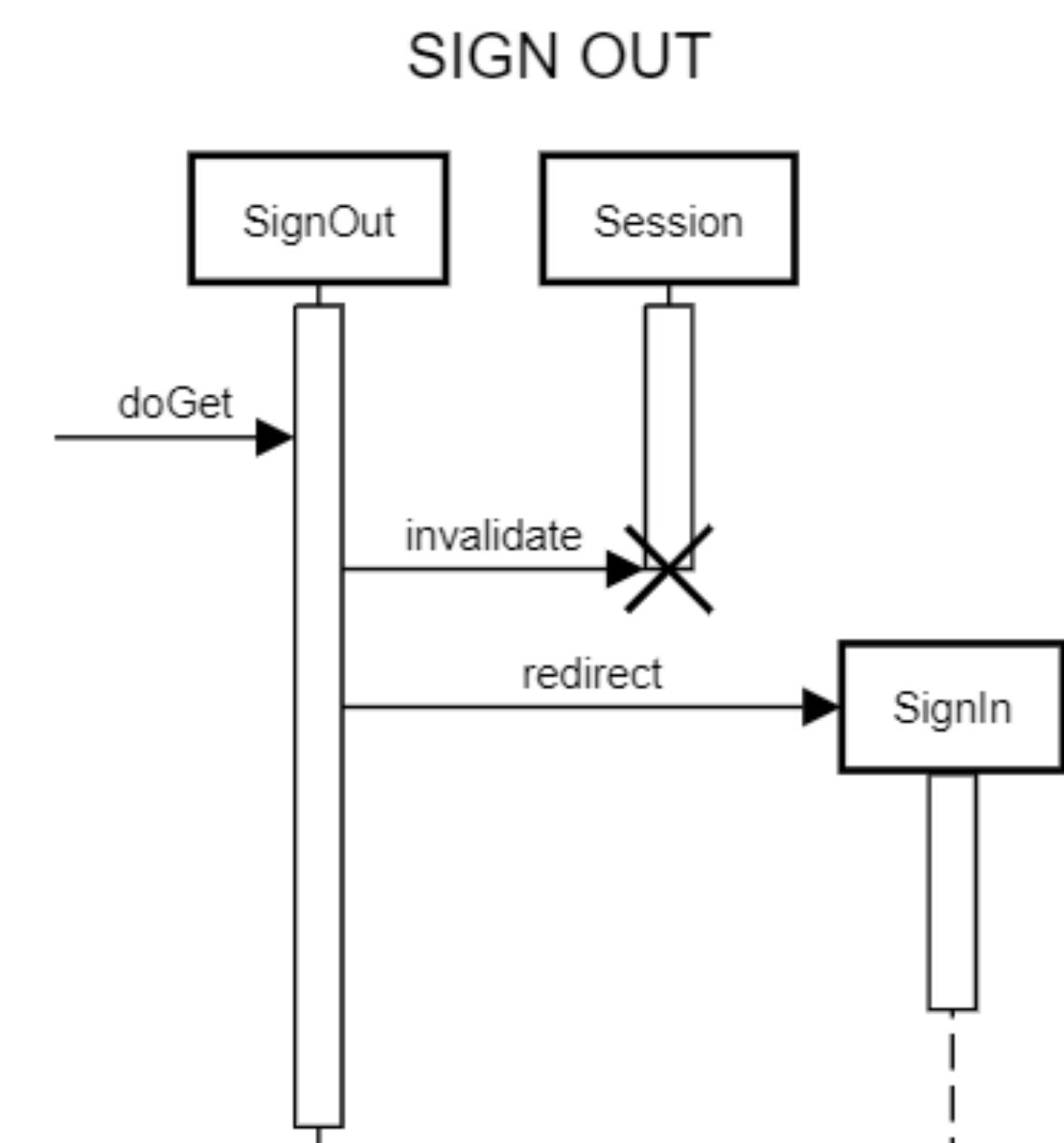
# Get Song



# Get Image



# Sign Out



# RIA

# ADDITIONAL SPEC

## ThePlaylist.html

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

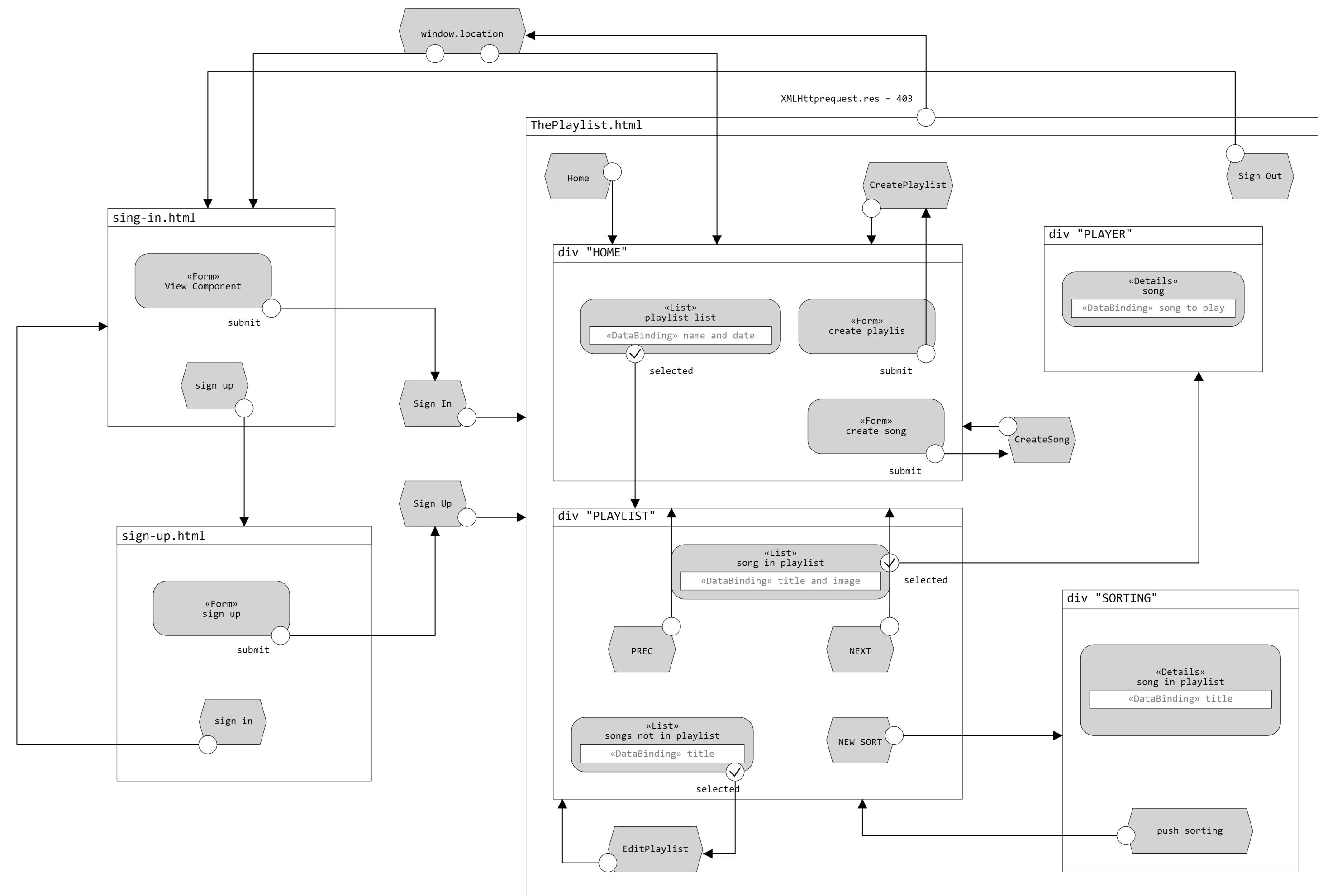
- Dopo il login dell'utente, l'intera applicazione è realizzata con un'unica pagina'. 
- Ogni interazione dell'utente è gestita senza ricaricare completamente la pagina, ma produce l'invocazione asincrona del server e l'eventuale modifica del contenuto da aggiornare a seguito dell'evento.
- L'evento 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 diverso da quello di default (data decrescente). Dalla HOME con un link associato a ogni playlist page si accede a una pagina RIORDINO, che mostra la lista completa dei brani della playlist e permette all'utente di 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.

# LOCLA DATABASE SCHEMA

- create table PLAYLIST(  
Name varchar(50) not null,  
UserName varchar(50) not null references  
USER(UserName) on delete cascade on  
update cascade,  
CreationDate date not null,  
Sorting json default null,  
primary key(Name, UserName)  
);

EVERYTHING ELSE REMAINED AS  
BEFORE

# APPLICATION DESIGN



# VIEWS AND COMPONENTS

MODEL OBJECTS (BEANS)

DATA ACCESS OBJECTS (DAO)

CONTROLLERS (SERVLET)

FILTERS AND UTILS

VIEWS (TEMPLATES)

SCRIPT (JS OBJECTS AND FUNCTIONS)

# MODEL OBJECTS

## BEANS

~ SONG

- ID, TITLE, GENRE, FILEAUDIO, ALBUM

~ ALBUM

- ID, TITLE, FILEIMAGE, SINGER, YEAR

~ PLAYLIST

- NAME, CREATIONDATE

# DATA ACCESS OBJECTS

## DAO

- PlaylistDAO:
  - AddPlaylist
  - SongAlreadyIn
  - GetSorting
  - + AddSongToPlaylist
  - + AddPlaylistWithSongs
  - + BelongTo
  - + AllPlaylist
  - + Taken
  - + EditSorting
  - + GetSongTitleAndImage

# DATA ACCESS OBJECTS

## DAO

- SongDAO:
  - FindAlbumId
  - AddAlbum
  - SongAlreadyIn
  - addSong
  - + AddSongAndAlbum
  - + GetSongByUser
  - + PlaySong
  - + PlaySong
  - + GetNumOfSongByUser
- UserDAO:
  - Taken
  - + Authentication
  - + Registration

# CONTROLLER

## SERVLET

- CreatePlaylistServlet
- CreateSongServlet
- EditPlaylistServlet
- EditSortingServlet
- GetPlaylistListServlet
- GetSongListServlet
- GetSongInPlaylist
- PlaySongServlet
- SignInServlet
- SignUpServlet
- SignOutServlet

# VIEWS

## TEMPLATE



# FILTERS AND UTILS

- LoggedChecker (403) • ConnectionHandler
- NotLoggedChecker (403) • FromJsonToArray
- PlayerChecker (400) • GetEncoding
- PlaylistChecker (400)

# JS SCRIPTS

- AddSongToPlaylist
- CreatePlaylist
- CreateSong
- DragAndDrop (manages the reordering of the sorting)
- Filter (manages the session and the correct redirect to sign-in.html)
- Render (manages what is shown by ThePlaylist.html)
- Reset (reset a “page” when it is not shown anymore)
- ServerCall (has makeCall function that makes the AJAX request)
- SignIn
- SignUp
- SignOut

# SCRIPT

## CONTROLLER / EVENT HANDLER

CLIENT		SERVER	
EVENT	CONTROLLER	EVENT	CONTROLLER
Sign In Submit	sign-in.js: makeCall	POST form	SignInServlet
Sign Up Submit	sign-up.js: makeCall	POST form	SignUpServlet
Window Load	render.js: showHomePage	GET	GetPlaylistListServlet
Window Load	render.js: showHomePage	GET	GetSongListServlet

# SCRIPT

## CONTROLLER / EVENT HANDLER

CLIENT		SERVER	
EVENT	CONTROLLER	EVENT	CONTROLLER
Home: createSong	createsong.js: makeCall	POST form	CreateSongServlet
Home: createPlaylist	createplaylist.js: makeCall	POST form	CreatePlaylistServlet
Home: selectPlaylist	render.js: showPlaylistPage	GET playlistName	GetSongsInPlaylistServlet
Playlist: NEXT / PREV	render.js: updateBlock	-	-

# SCRIPT

## CONTROLLER / EVENT HANDLER

CLIENT		SERVER	
EVENT	CONTROLLER	EVENT	CONTROLLER
Playlist: updatePlaylist	render.js: updatePlaylist	POST form	EditPlaylistServlet
Playlist: newSorting	render.js: showSortingPage	-	-
Sorting: drag and drop	dragANDdrop.js	-	-
Sorting: pushSorting	drag&drop.js: pushSorting	POST jsonObject	EditSortingServlet

# SCRIPT

## CONTROLLER / EVENT HANDLER

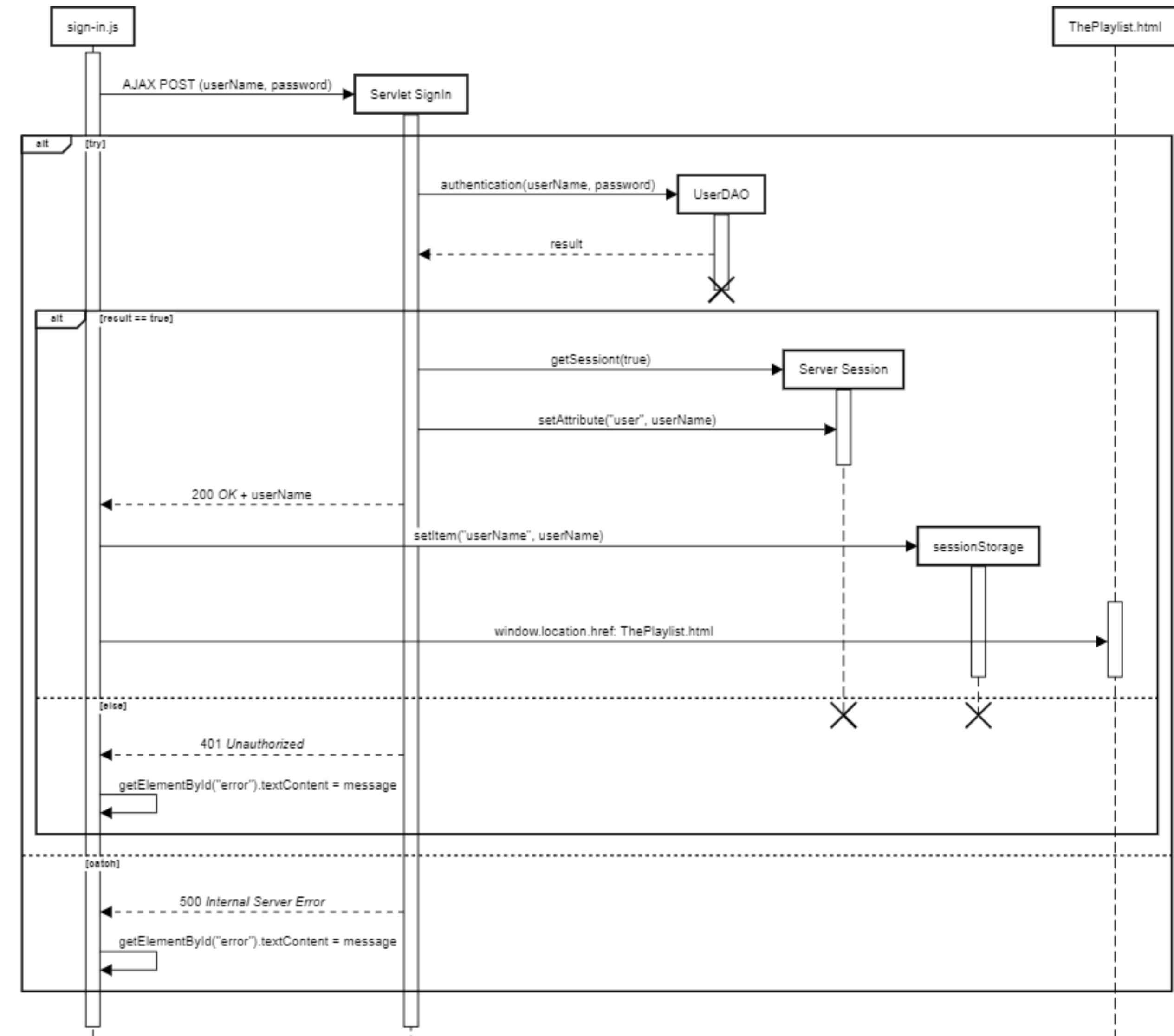
CLIENT		SERVER	
EVENT	CONTROLLER	EVENT	CONTROLLER
Playlist: selectSong	render.js: showPlayerPage	POST songId	PlaySongServlet
GoHome	render.js: getHome	-	-
Sign Out	sign-out.js: makeCall	GET	SignInServlet

# SEQUENCE DIAGRAM

**every DAO constructor has the database connection as parameter**

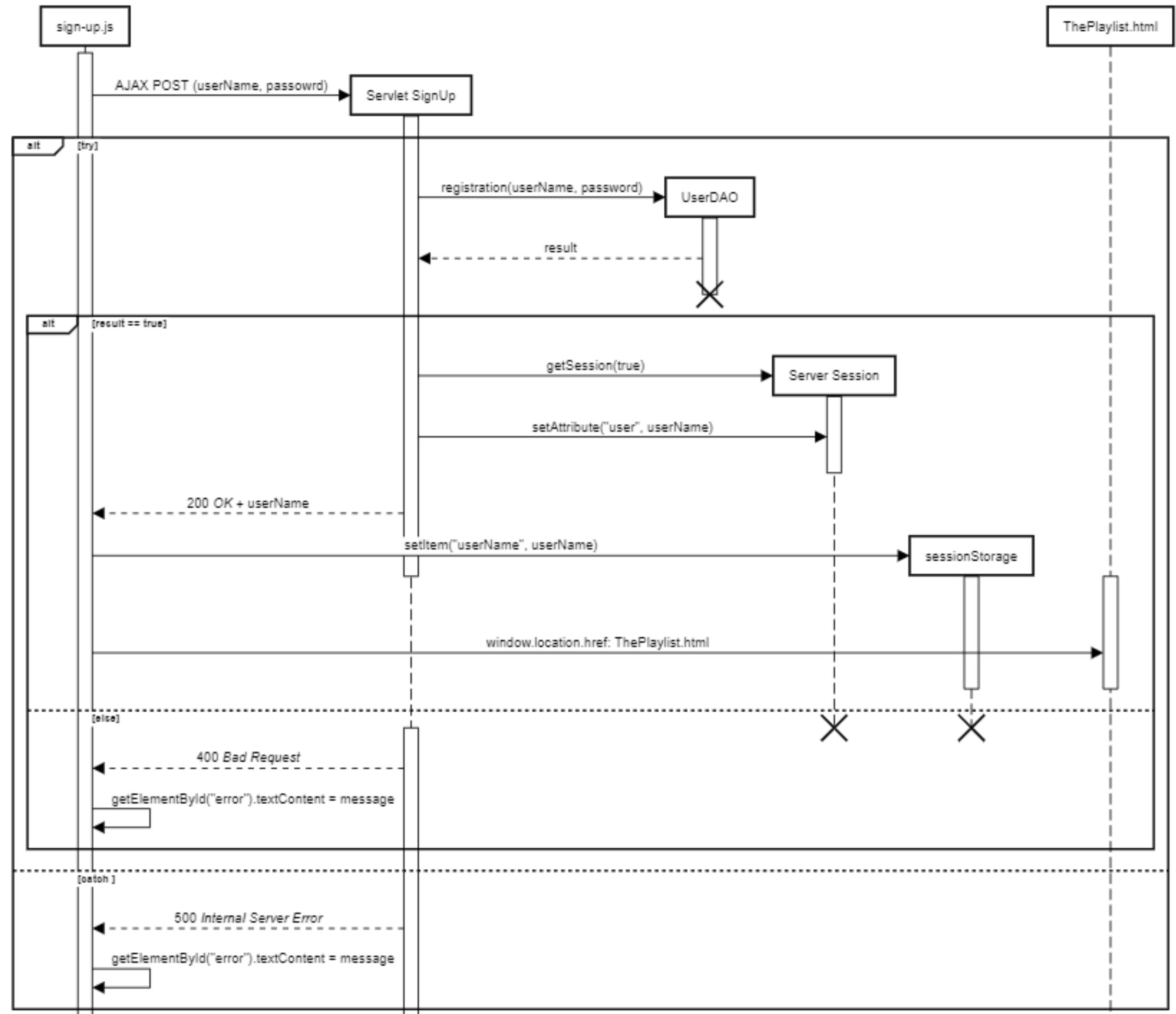
# Sign In

SIGN IN



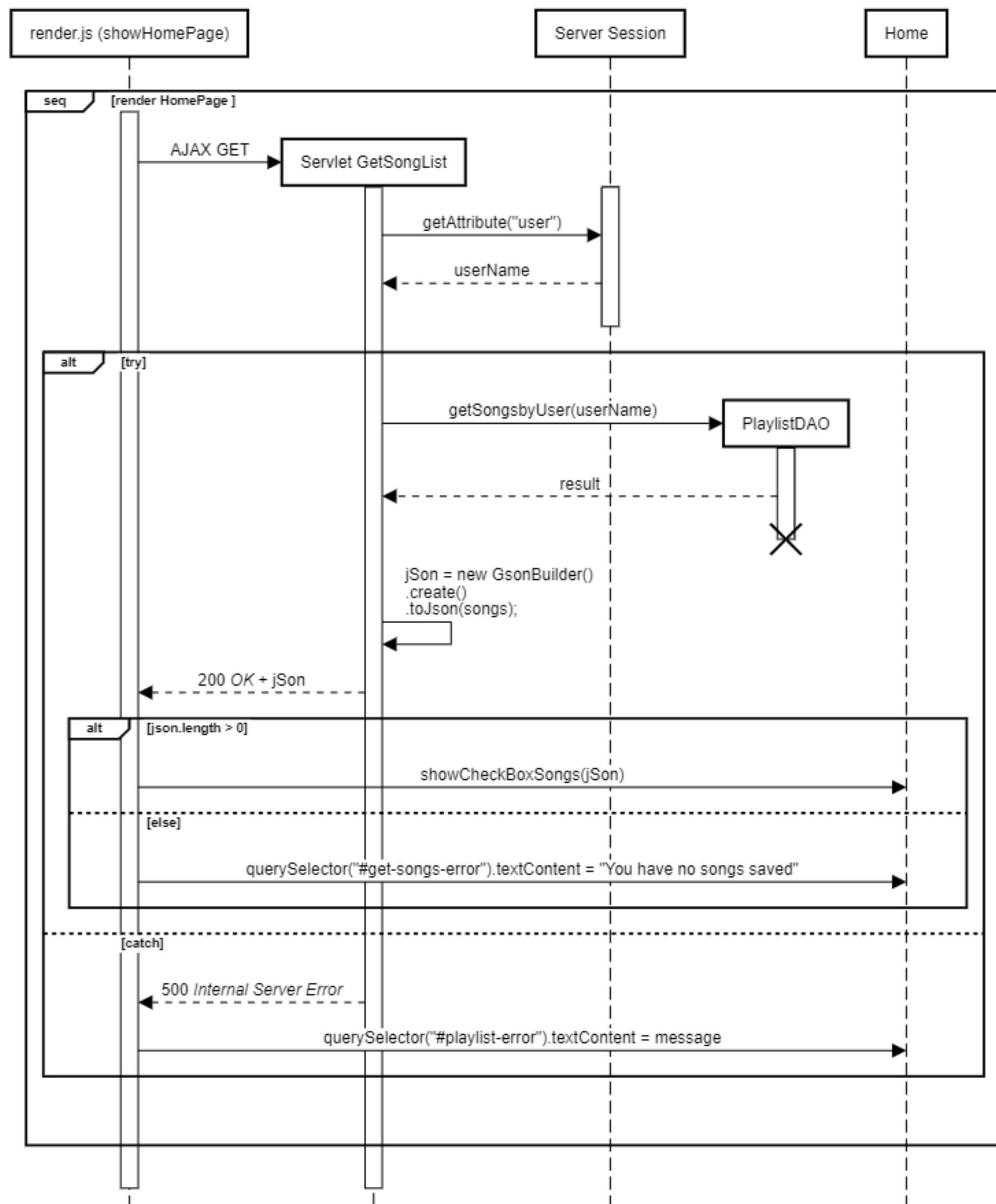
# Sign Up

SIGN UP



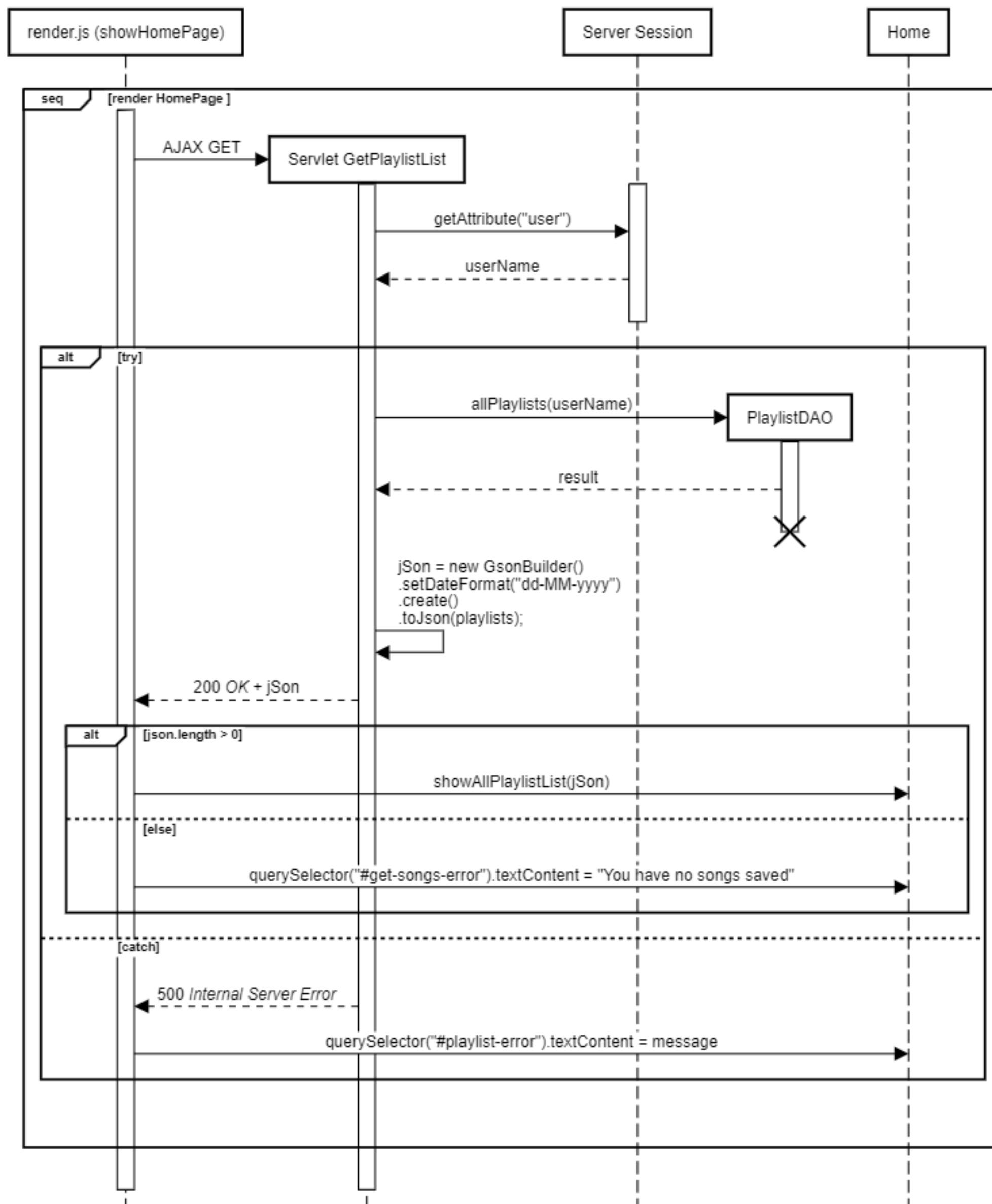
# Song List

SONG LIST

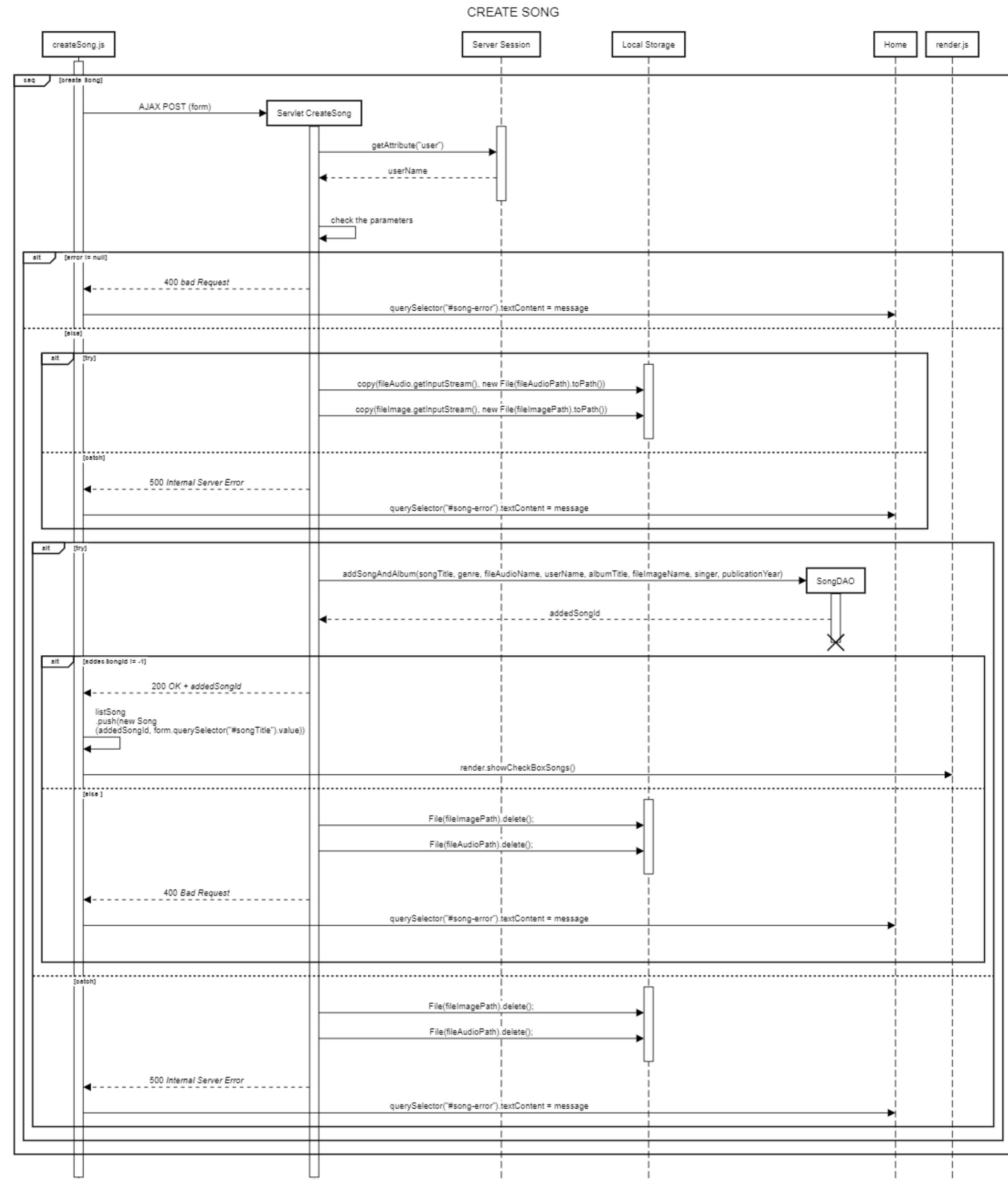


# Playlist List

## PLAYLIST LIST

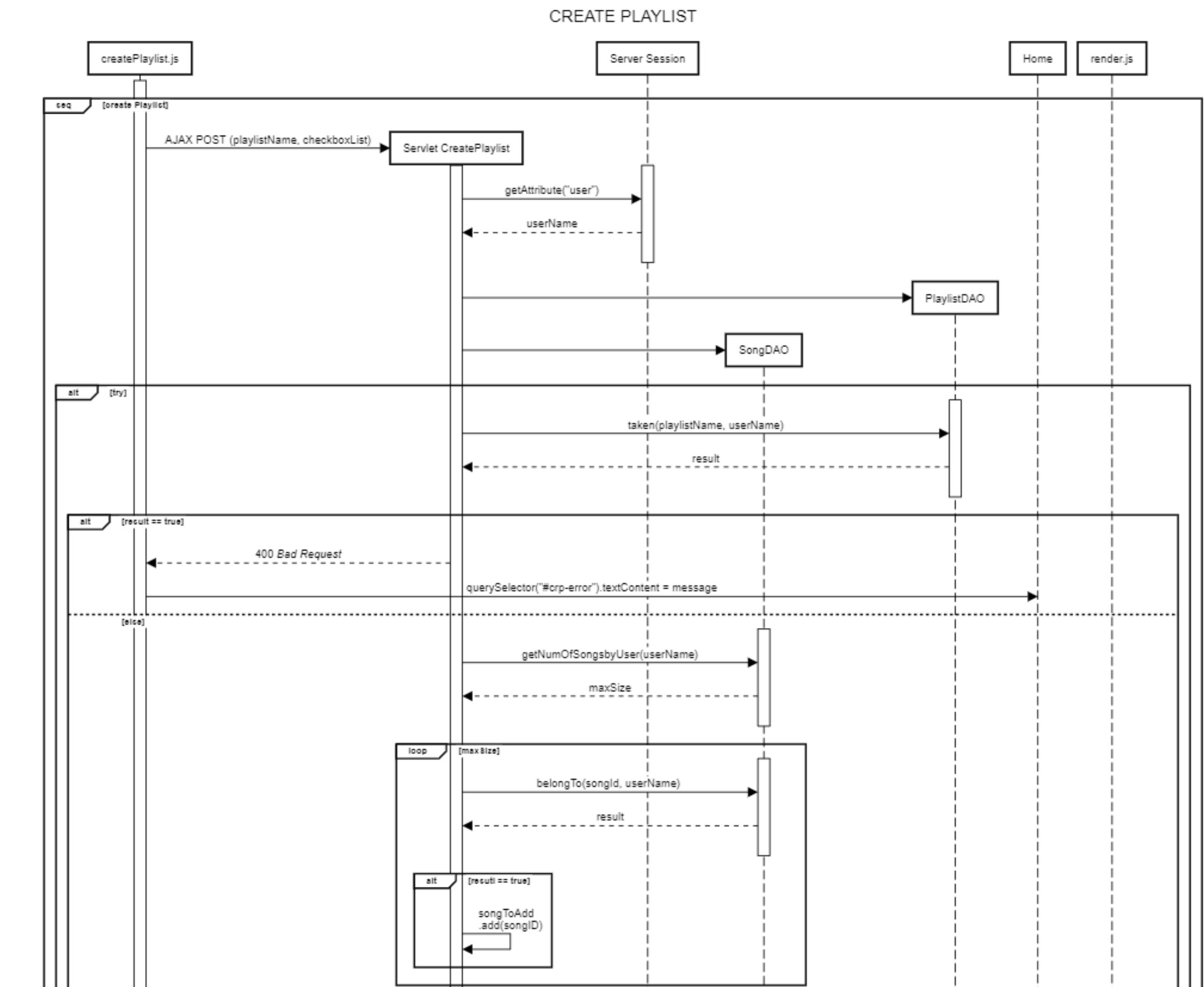


# Create Song

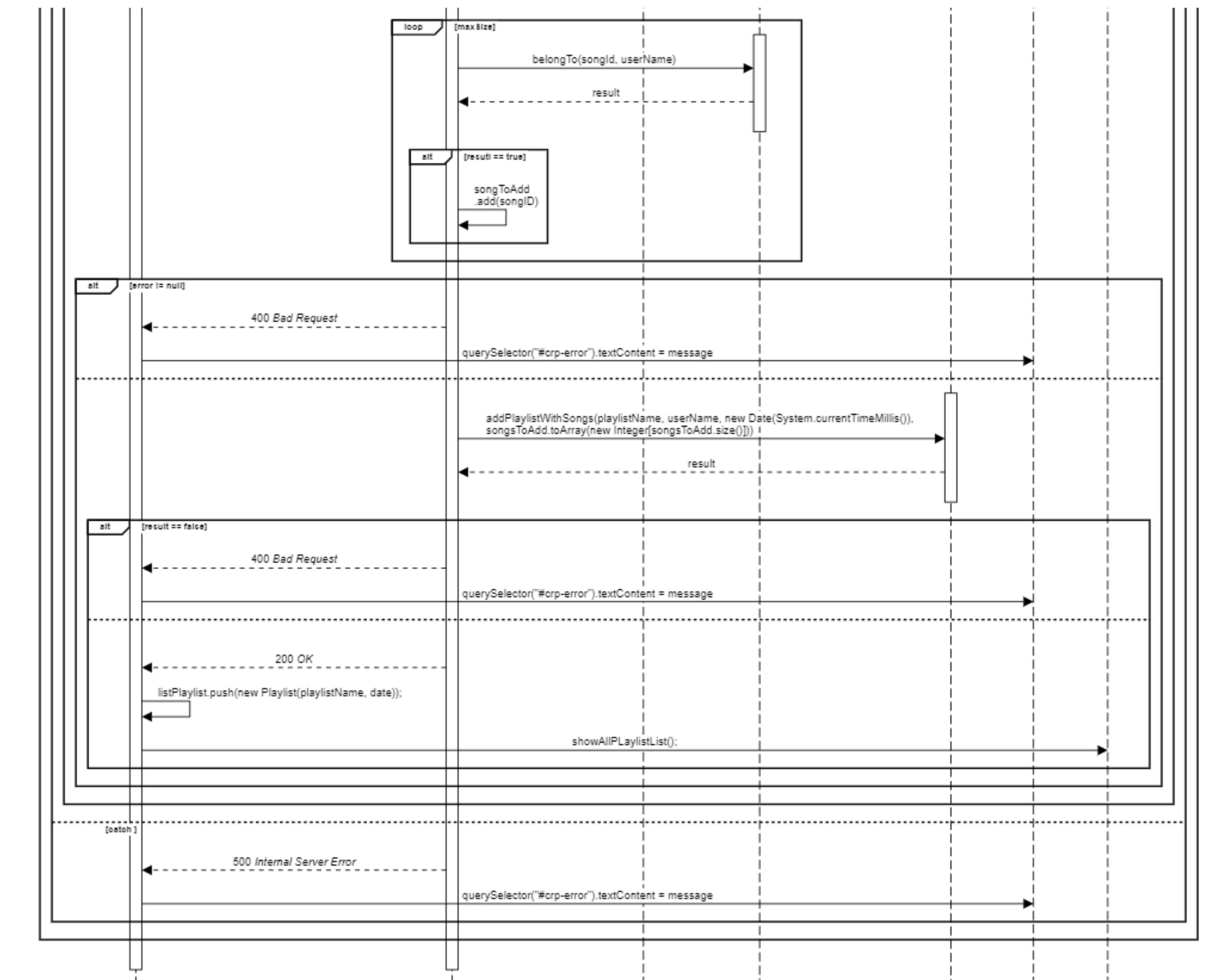


# Create Playlist

GO NEXT PAGE

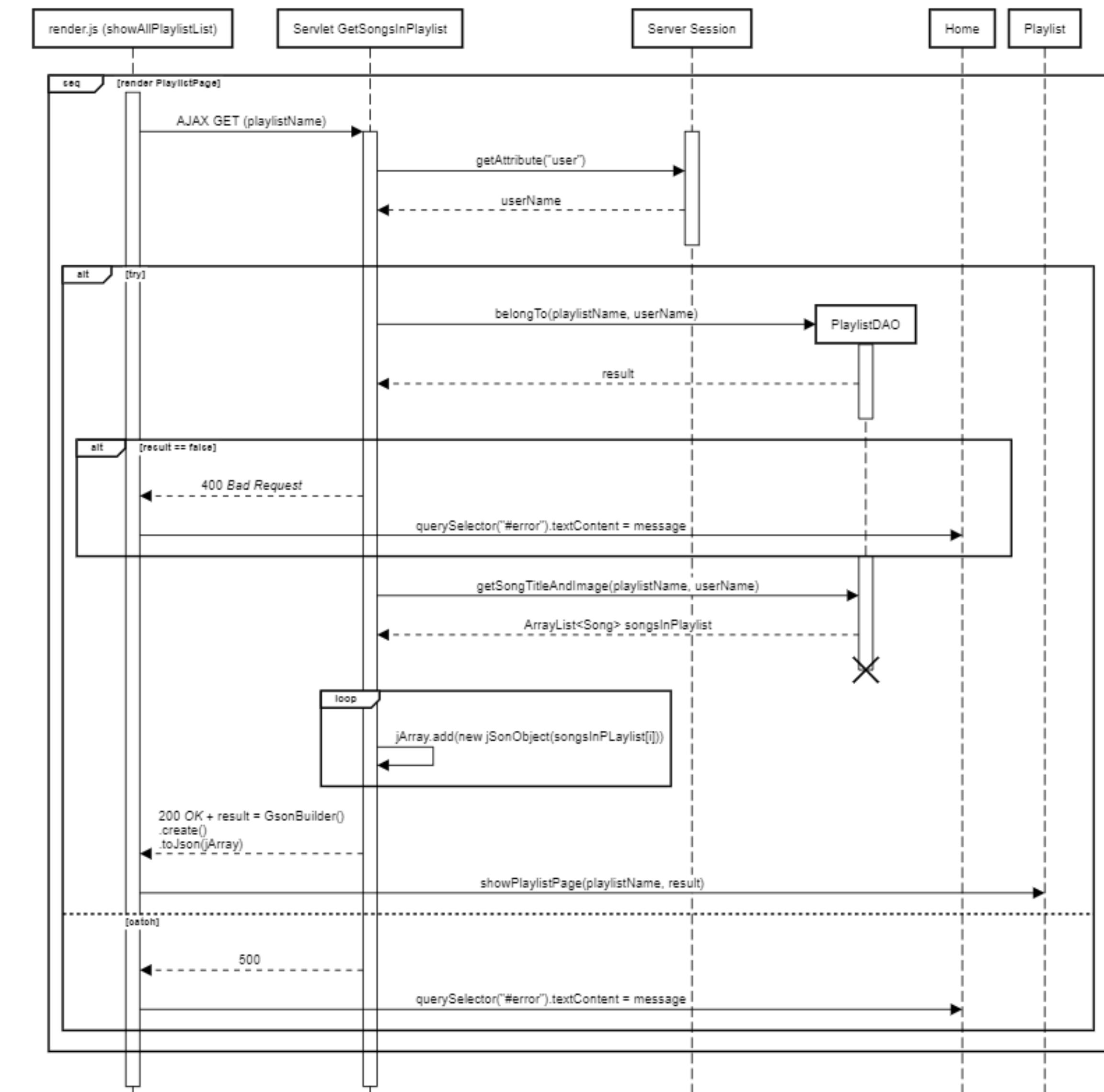


# Create Playlist



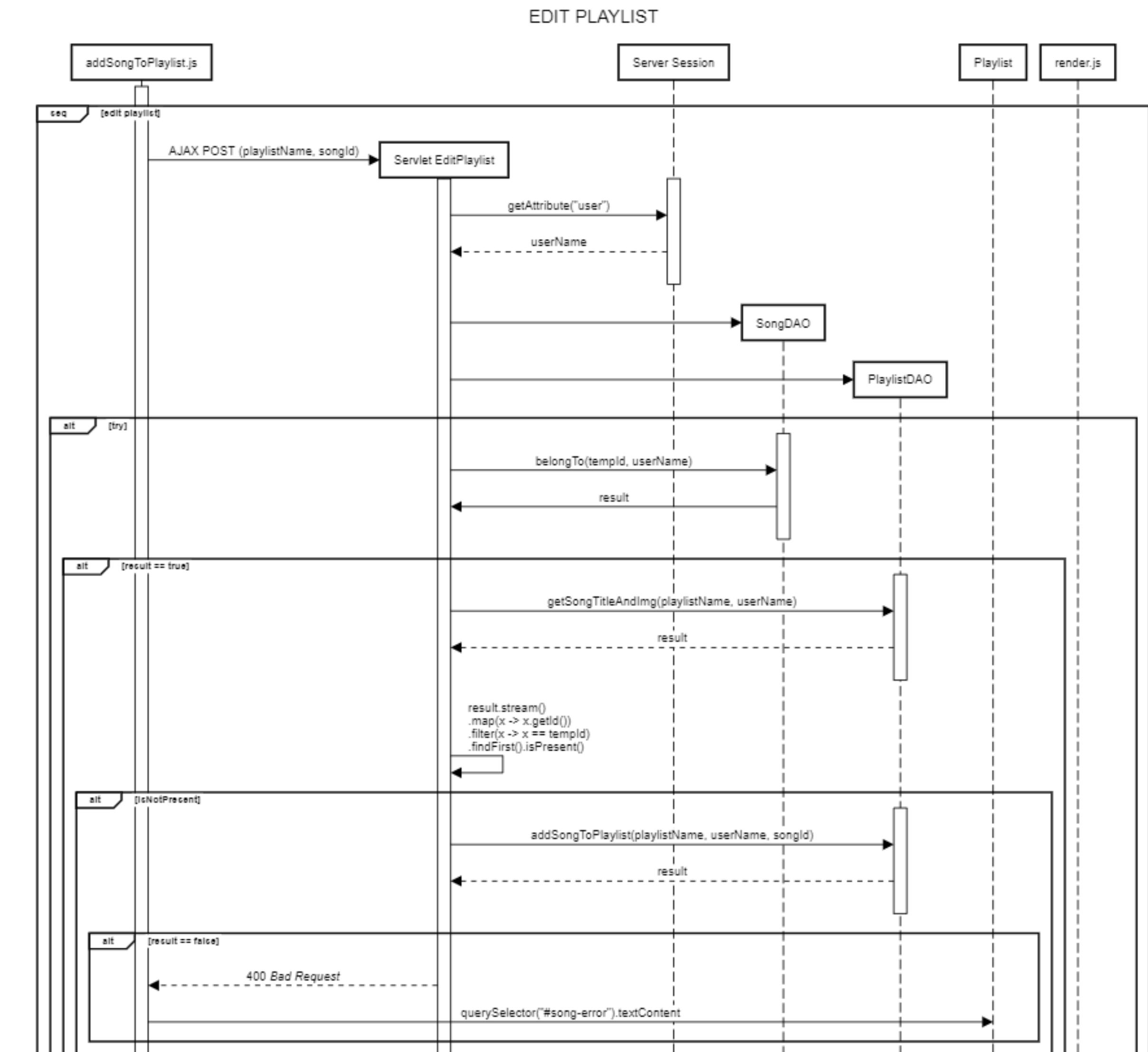
# Songs In Playlist

## SONGS IN PLAYLIST

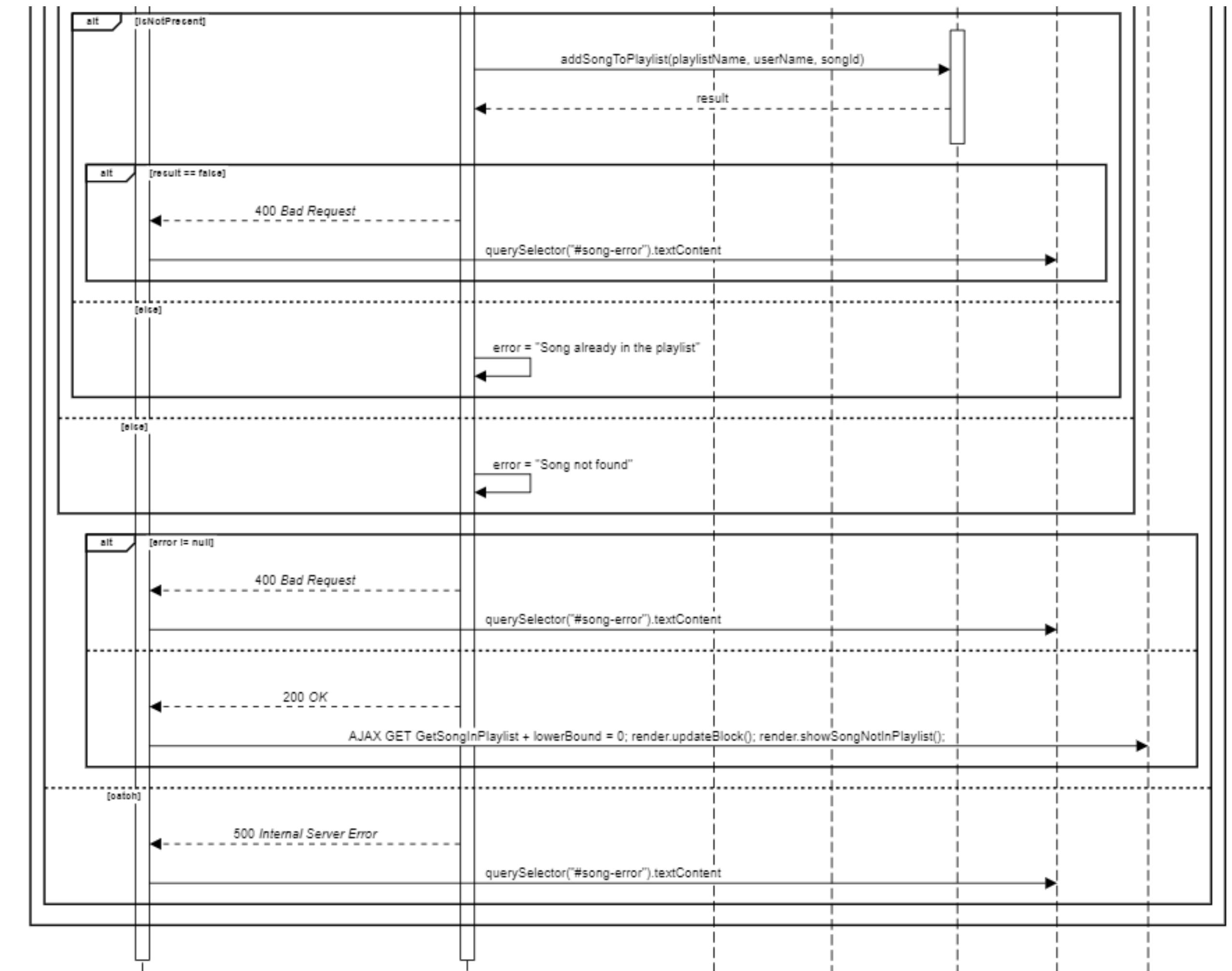


# Edit Playlist

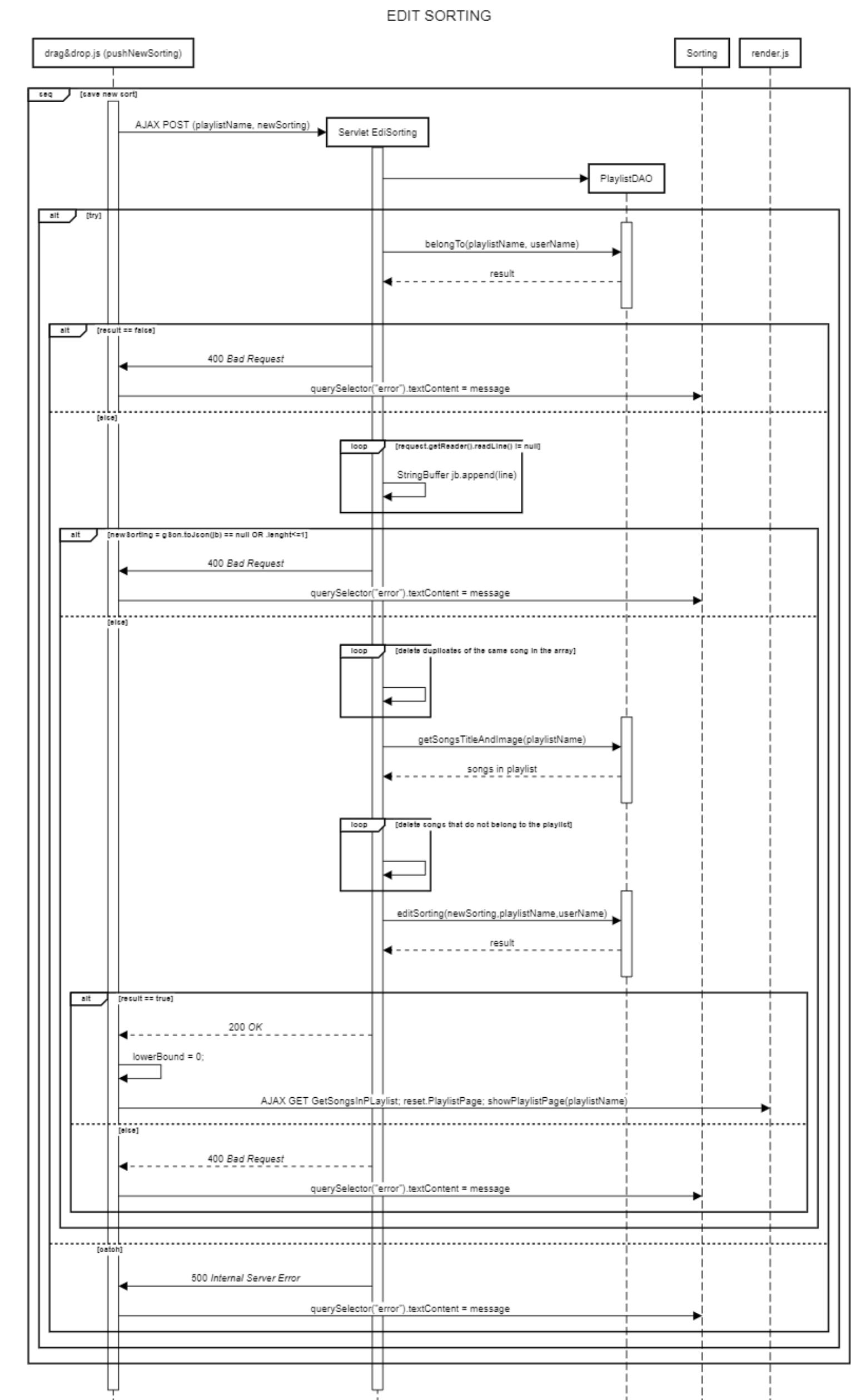
GO NEXT PAGE



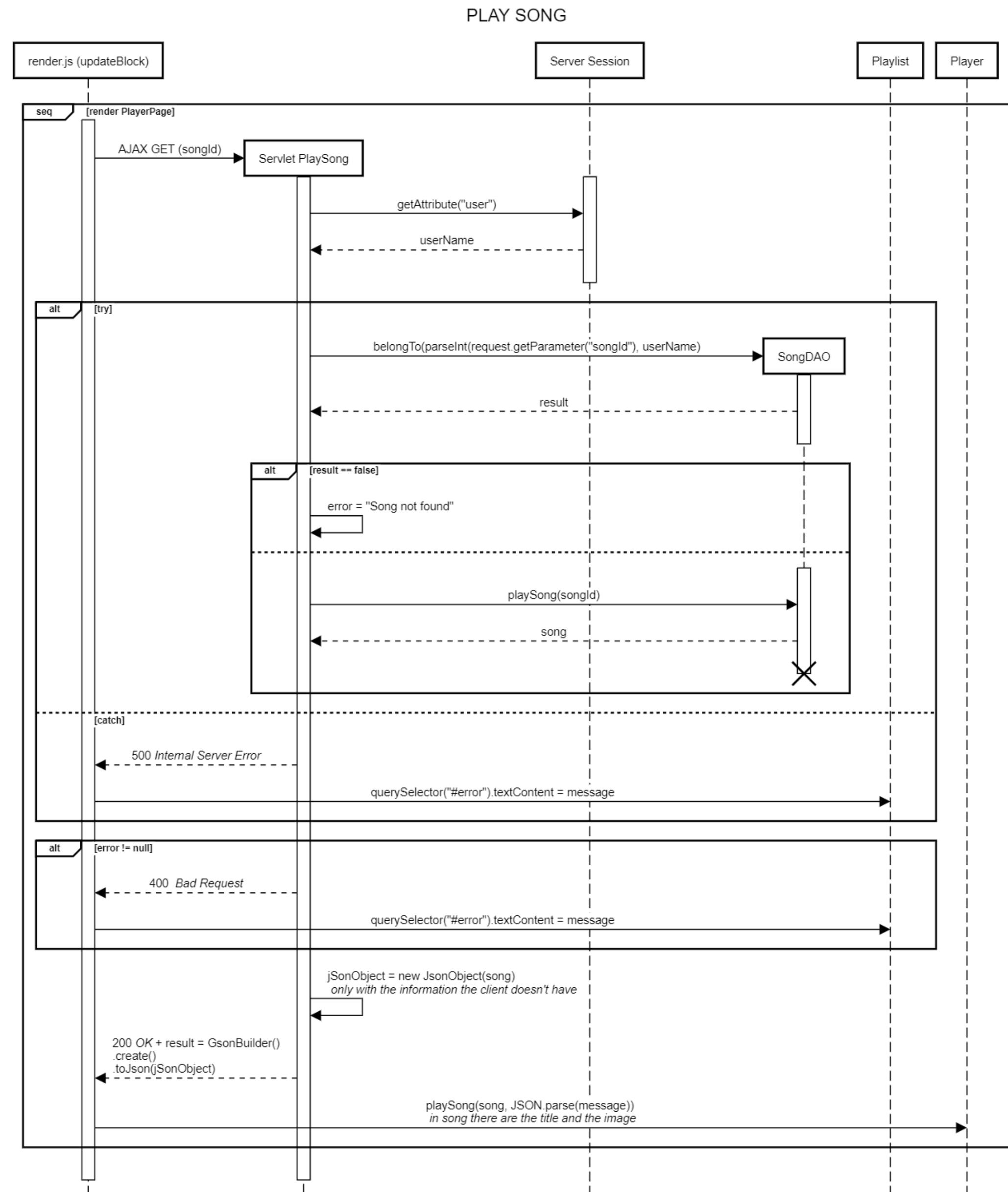
# Edit Playlist



# Edit Sorting

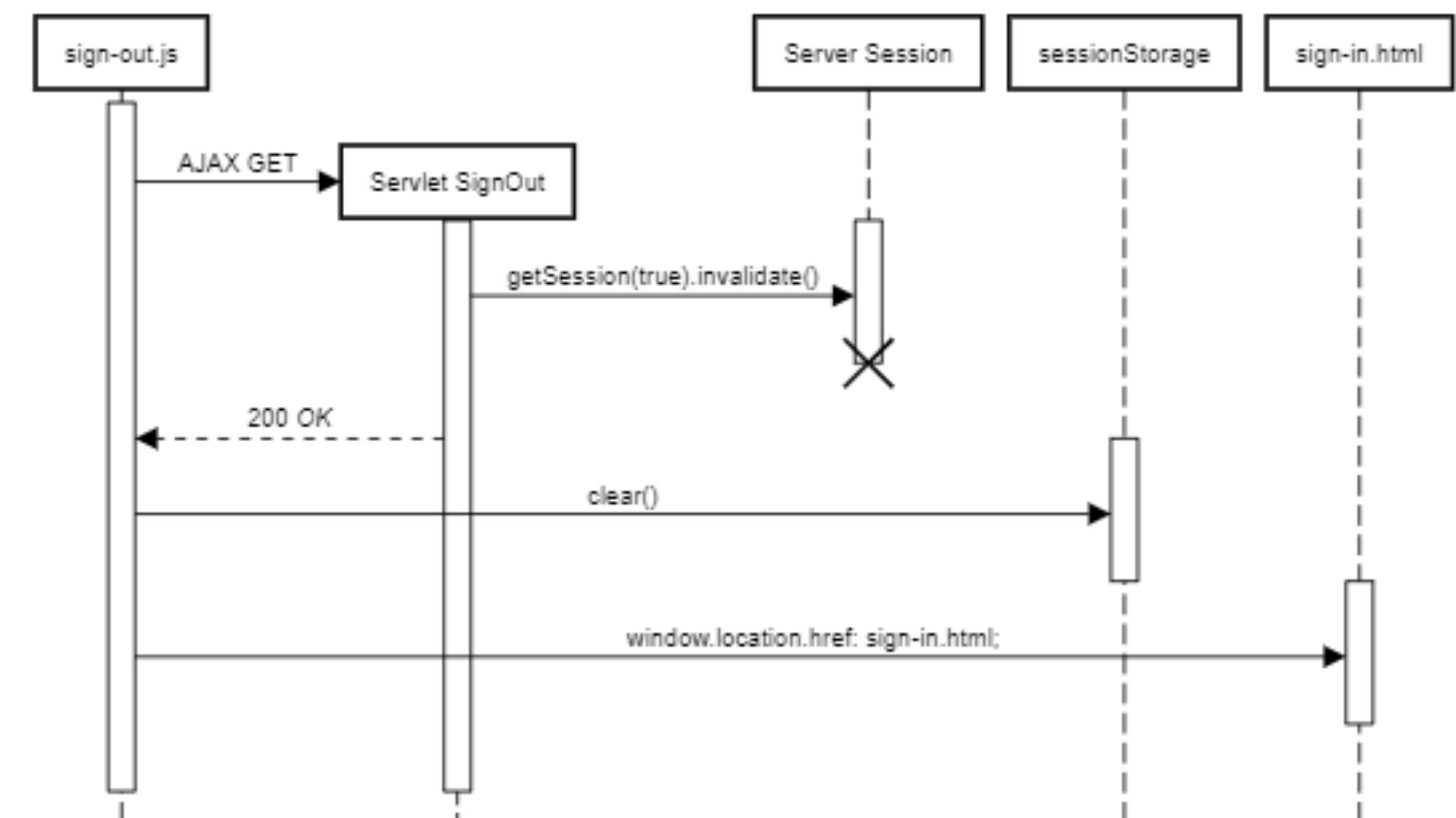


# Play Song



# Sign Out

SIGN OUT



**END**