Playlist Musicale
Versione Javascript

Web Technologies

Melanie Tonarelli

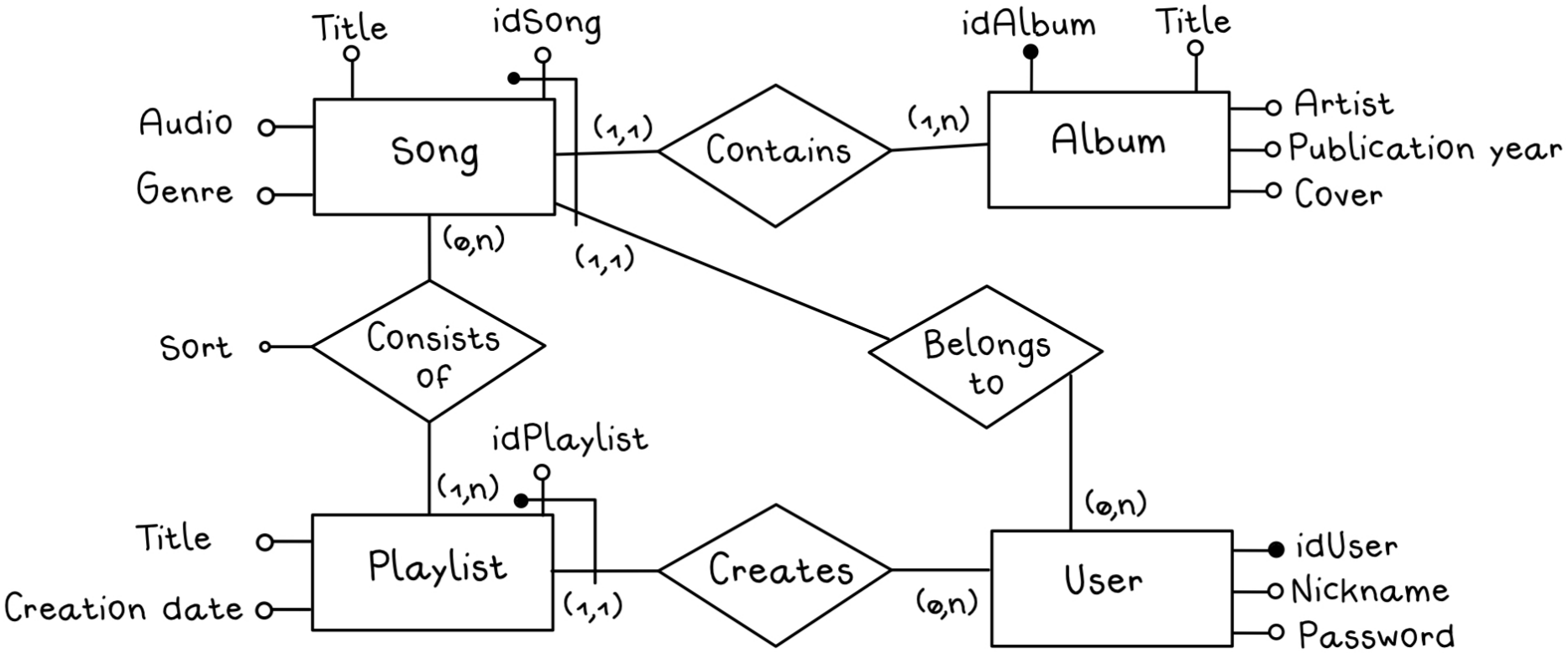
10787497

Playlist Musicali

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.

DB design


Title
idSong
idAlbum
Title
Audio
Genre
Song
(1,1)
Contains
(1,n)
Album
Artist
Publication year
Cover
(0,n)
(1,1)
Sort
Consists of
Belongs to
idPlaylist
(1,n)
Title
Creation date
Playlist
(1,1)
Creates
(0,n)
User
(0,n)
idUser
Nickname
Password

Local database schema

```
CREATE TABLE `Album` (
  `idAlbum` int NOT NULL AUTO_INCREMENT,
  `titleAlbum` varchar(45) NOT NULL,
  `artist` varchar(45) NOT NULL,
  `year` year NOT NULL,
  `cover` longblob NOT NULL,
  `userId` varchar(45) NOT NULL,
  PRIMARY KEY (`idAlbum`),
  UNIQUE KEY `constraint_unique`
        (`titleAlbum`,`artist`,`userId`),
  KEY `user_idx` (`userId`),
  CONSTRAINT `userId` FOREIGN KEY (`userId`) REFERENCES `User`
      (`username`) ON UPDATE CASCADE
)
```

Local database schema

```
CREATE TABLE `InPlaylist` (
  `playlist` int NOT NULL,
  `song` int NOT NULL,
  `sort` int NOT NULL DEFAULT '0',
  PRIMARY KEY (`playlist`,`song`),
  KEY `song_idx` (`song`),
  CONSTRAINT `playlist` FOREIGN KEY (`playlist`) REFERENCES
     `Playlist` (`idPlaylist`) ON UPDATE CASCADE,
  CONSTRAINT `song` FOREIGN KEY (`song`) REFERENCES `Song`
     (`idSong`) ON UPDATE CASCADE
)

CREATE TABLE `User` (
  `username` varchar(45) NOT NULL,
  `password` varchar(45) NOT NULL,
  `name` varchar(45) NOT NULL,
   PRIMARY KEY (`username`)
)
```

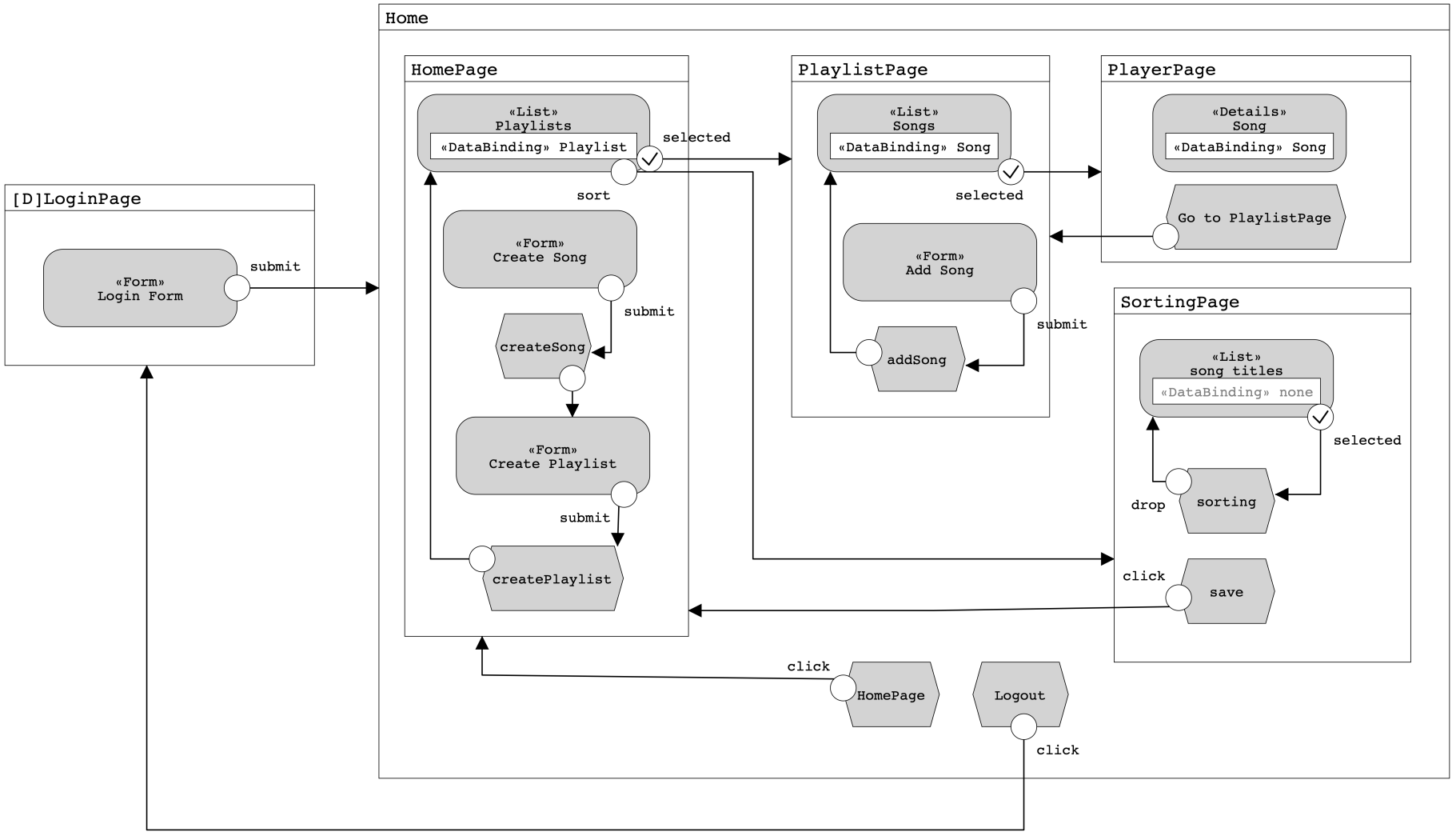
Local database schema

```
CREATE TABLE `Song` (
  `idSong` int NOT NULL AUTO_INCREMENT,
  `user` varchar(45) NOT NULL,
  `album` int NOT NULL,
  `title` varchar(45) NOT NULL,
  `file` longblob,
  `genre` varchar(45) NOT NULL,
  PRIMARY KEY (`idSong`),
  UNIQUE KEY `constraint_unique` (`user`,`album`,`title`),
  KEY `user_idx` (`user`),
  KEY `album_idx` (`album`),
  CONSTRAINT `album` FOREIGN KEY (`album`) REFERENCES
      `Album` (`idAlbum`) ON UPDATE CASCADE,
  CONSTRAINT `user` FOREIGN KEY (`user`) REFERENCES `User`
      (`username`) ON UPDATE CASCADE
)
```

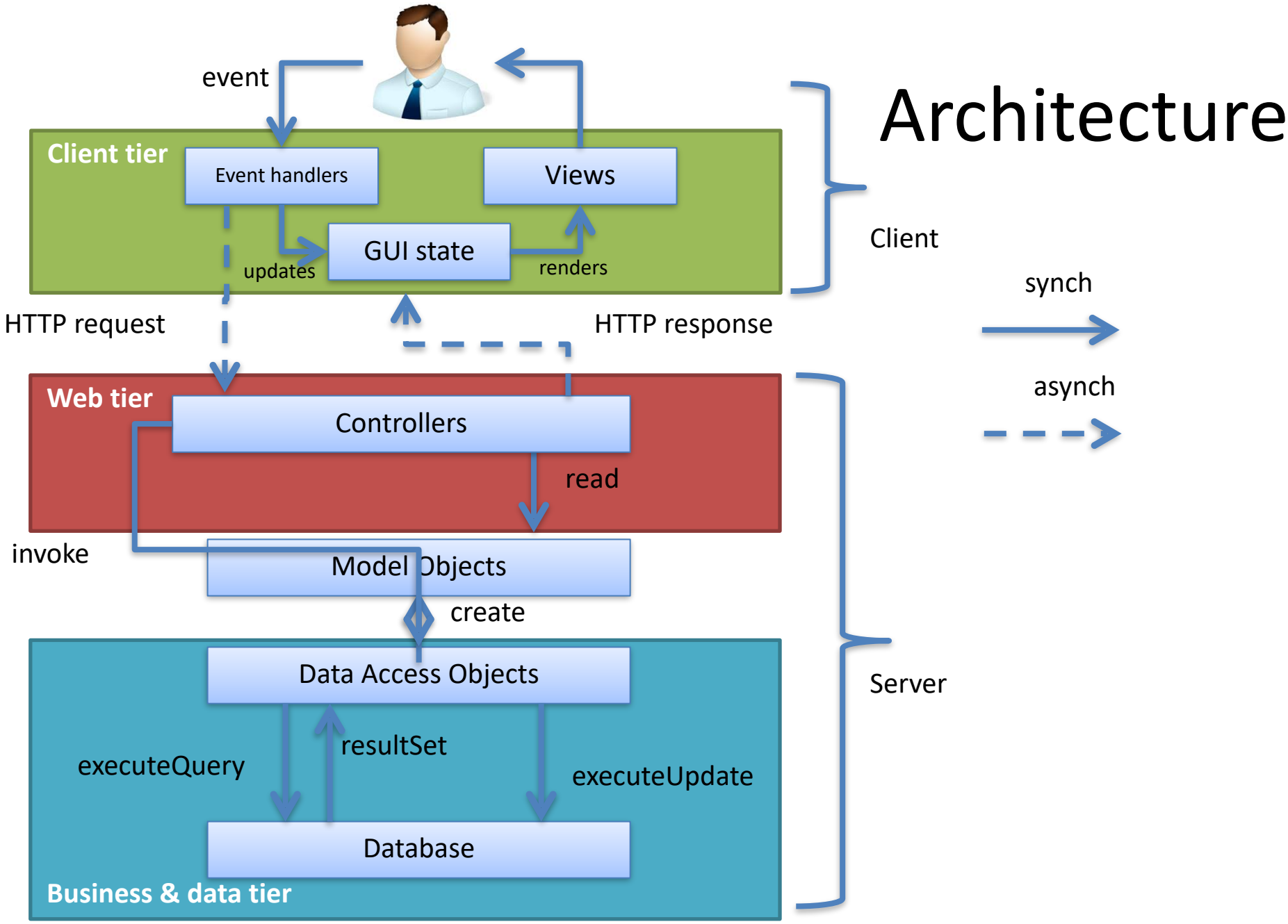
Local database schema

```
CREATE TABLE `Playlist` (
  `idPlaylist` int NOT NULL AUTO_INCREMENT,
  `idUser` varchar(45) NOT NULL,
  `name` varchar(45) NOT NULL,
  `creationDate` datetime NOT NULL DEFAULT
     CURRENT_TIMESTAMP,
  PRIMARY KEY (`idPlaylist`),
  UNIQUE KEY `constraint_unique` (`idUser`,`name`),
  KEY `idUser_idx` (`idUser`),
  CONSTRAINT `idUser` FOREIGN KEY (`idUser`) REFERENCES
     `User` (`username`) ON UPDATE CASCADE
)
```

Application design (in IFML)



Architecture



Components

- Model objects (Beans)
  - User
  - Playlist
  - Album
  - Song
- Filters
  - LoginChecker
- View
  - Home.html
  - LoginPage.html

- Controllers
  - AddSelectedSorting
  - AddSong
  - CreatePlaylist
  - CreateSong
  - GetPlaylists
  - GetSong
  - GetSongs
  - GetSongsInPlaylist
  - GetSongsNotInPlaylist
  - Login
  - Logout

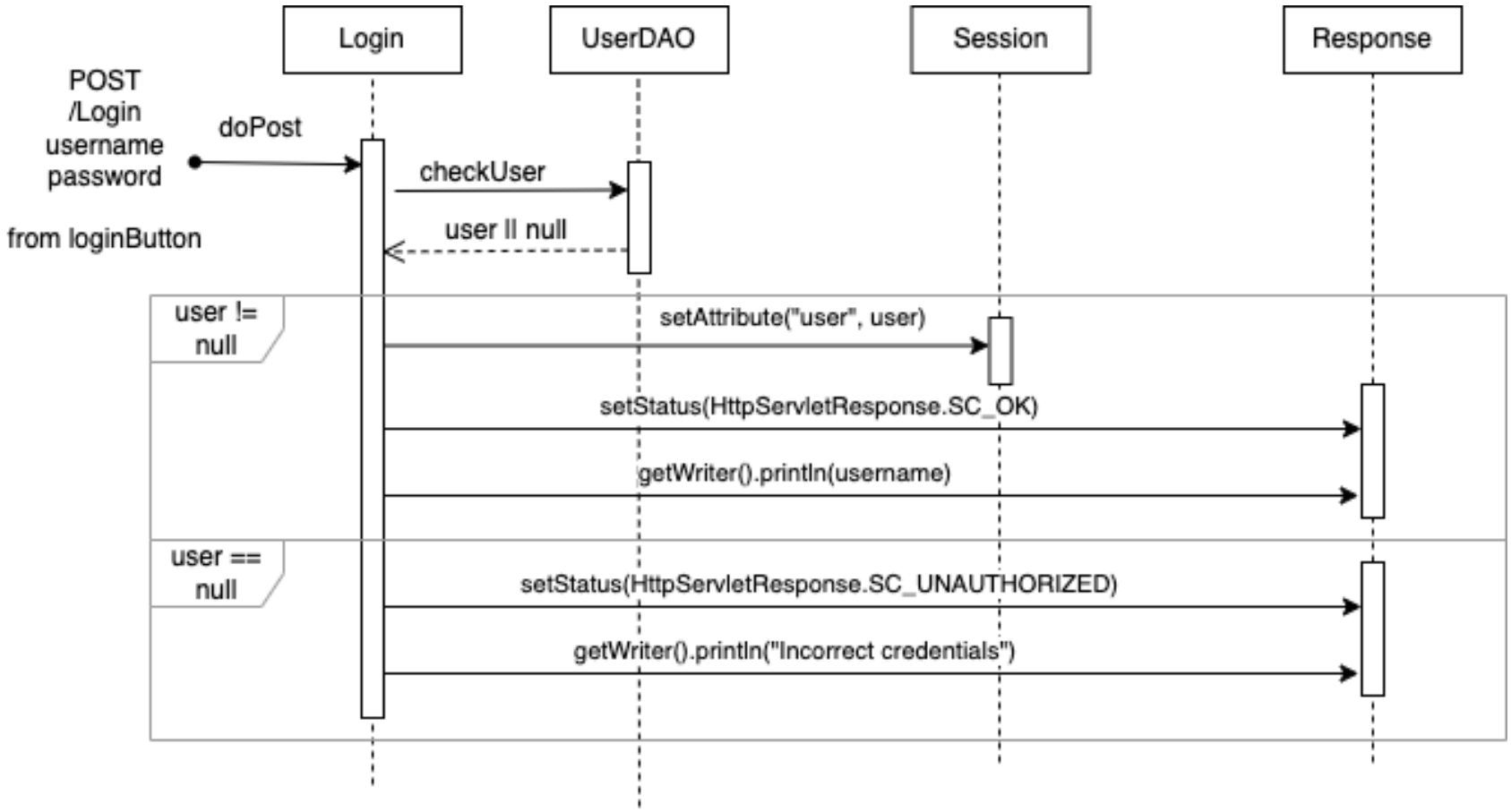
Component: Data Access Object

- UserDAO
  - checkUser(username, password)
- PlaylistDAO
  - findPlaylists(username)
  - createPlaylistWithSongs(username, titlePlaylist, songs)
  - -createPlaylist(username, titlePlaylist)
  - -addSongInPlaylist(idPlaylist, idSong)
  - -findIdPlaylist(username, title)
  - isPlaylistPresent(username, idPlaylist)
  - getPlaylistById(username. idPlaylist)
  - isSongPresentInPlaylist(idSong, idPlaylist)
  - addSorting(idPlaylist, selectedSorting)
  - getSorting(idPlaylist)
- SongDAO
  - createSongAlbum(user, titleSong, genre, file, titleAlbum, artist, year, cover)
  - -createSong(user, title, genre, file, idAlbum)
  - -findIdAlbum(user, title, artist)
  - -createAlbum(user, title, artist, year, cover)
  - findAllSongsByUsername(username)
  - findAllSongsInPlaylist(username, idPlaylist)
  - findSongsNotInPlaylist(username, idPlaylist)
  - findAllSongInfoById(username, idSong)
  - isSongPresent(username, idSong)
  - songAlreadyExists(username, titleSong, titleAlbum, artist, year, cover)

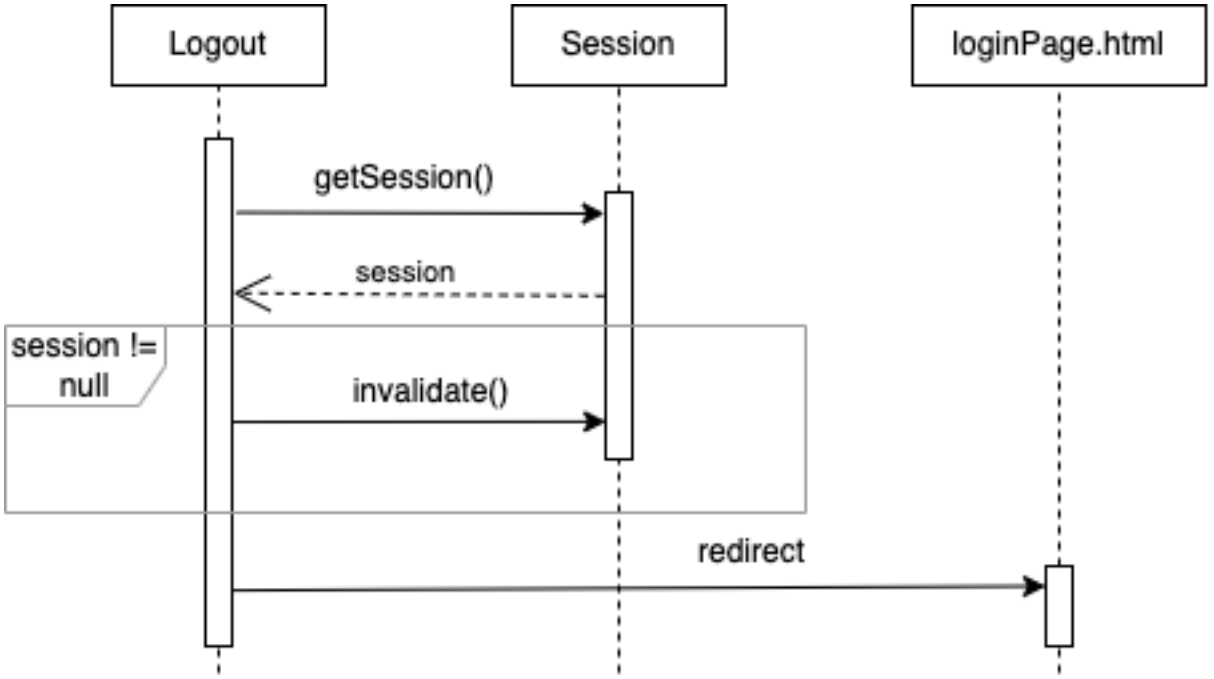
Sequence Diagram

- Per motivi di chiarezza e semplicità, ometto i controlli sui parametri in ingresso e la loro relativa gestione.
- Ogni volta che c'è un'istanza di un oggetto DAO, essa riceve come parametro una connection al database.
- Ogni richiesta al server successiva al login passa attraverso il filtro 'LoginChecker' il quale verifica se la sessione è valida e se l'utente è loggato, ovvero se l'attributo 'user' nella request non è null

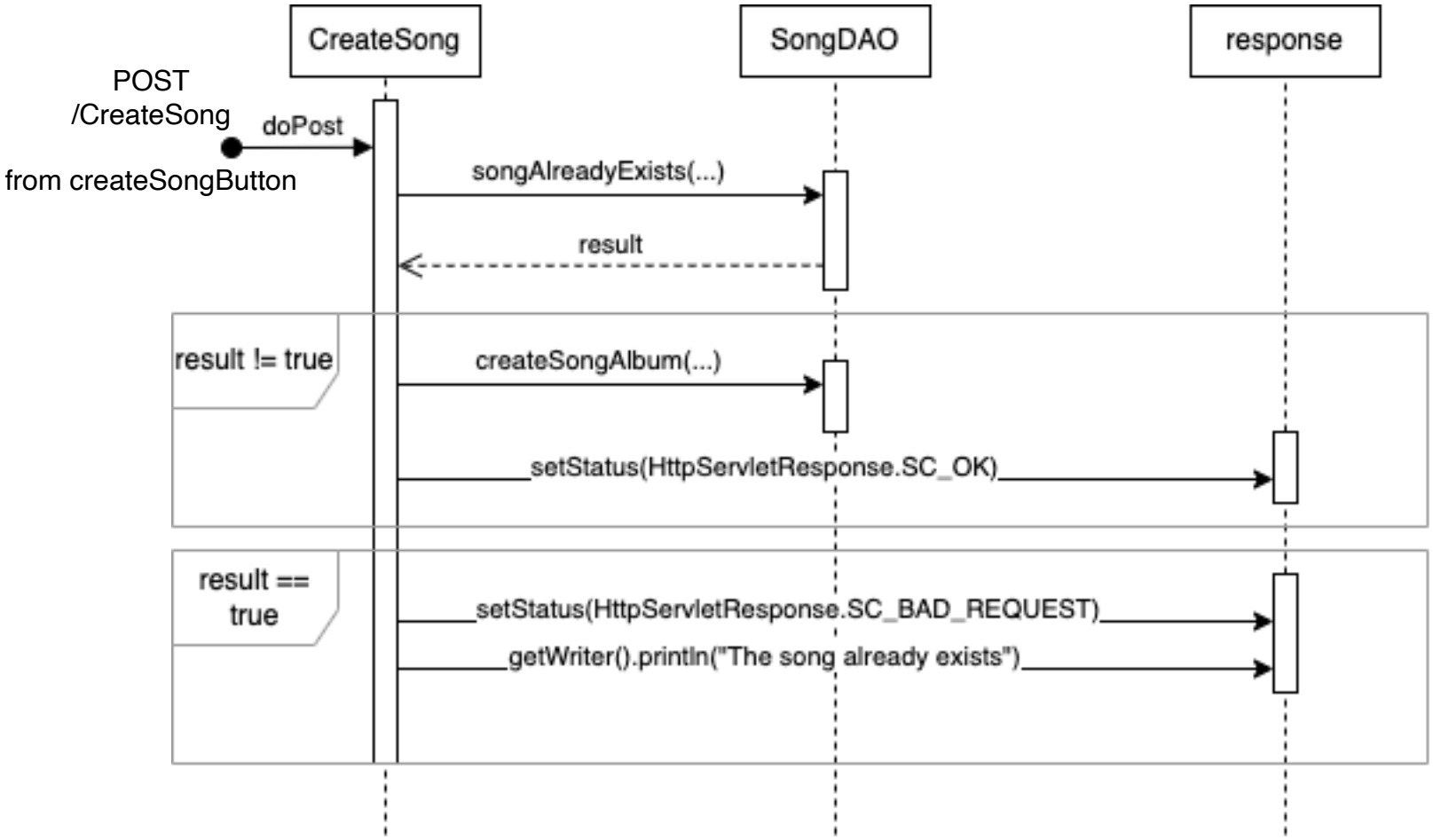
Event: login


Login
UserDAO
Session
Response
POST
/Login
username
password
doPost
checkUser
user II null
from loginButton
user != null
setAttribute("user", user)
setStatus(HttpServletResponse.SC_OK)
getWriter().println(username)
user == null
setStatus(HttpServletResponse.SC_UNAUTHORIZED)
getWriter().println("Incorrect credentials")

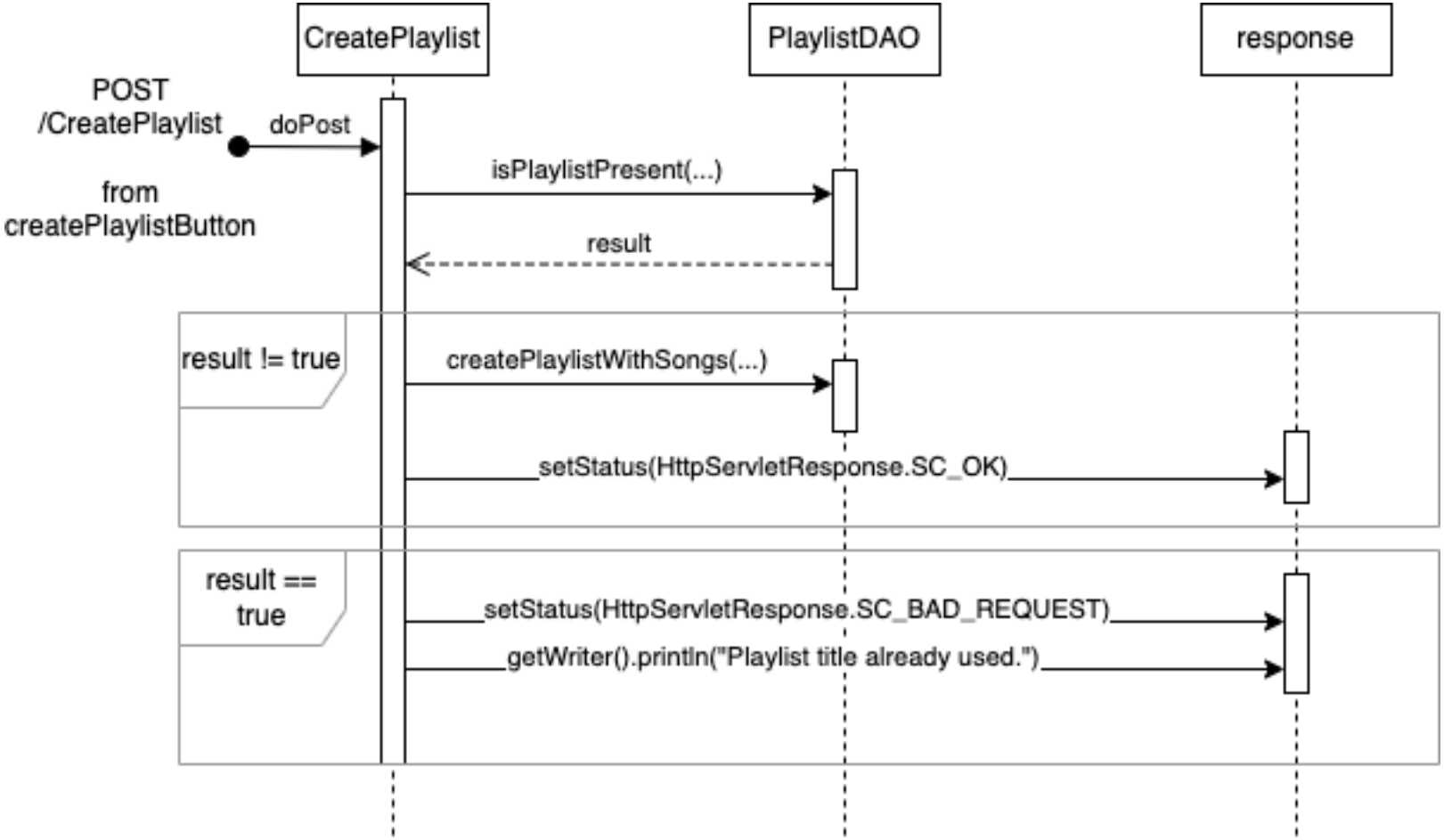
Event: logout



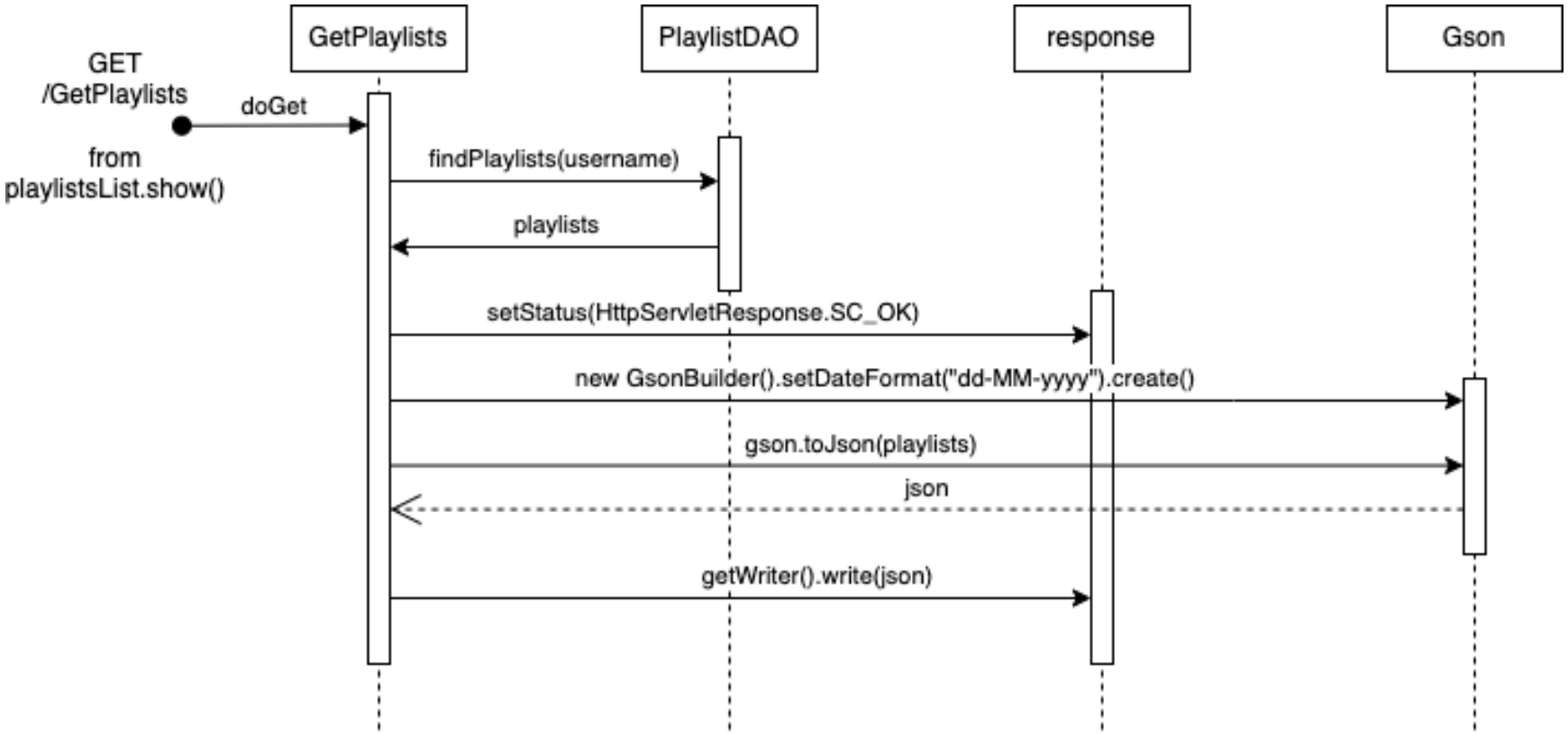
Event: create song



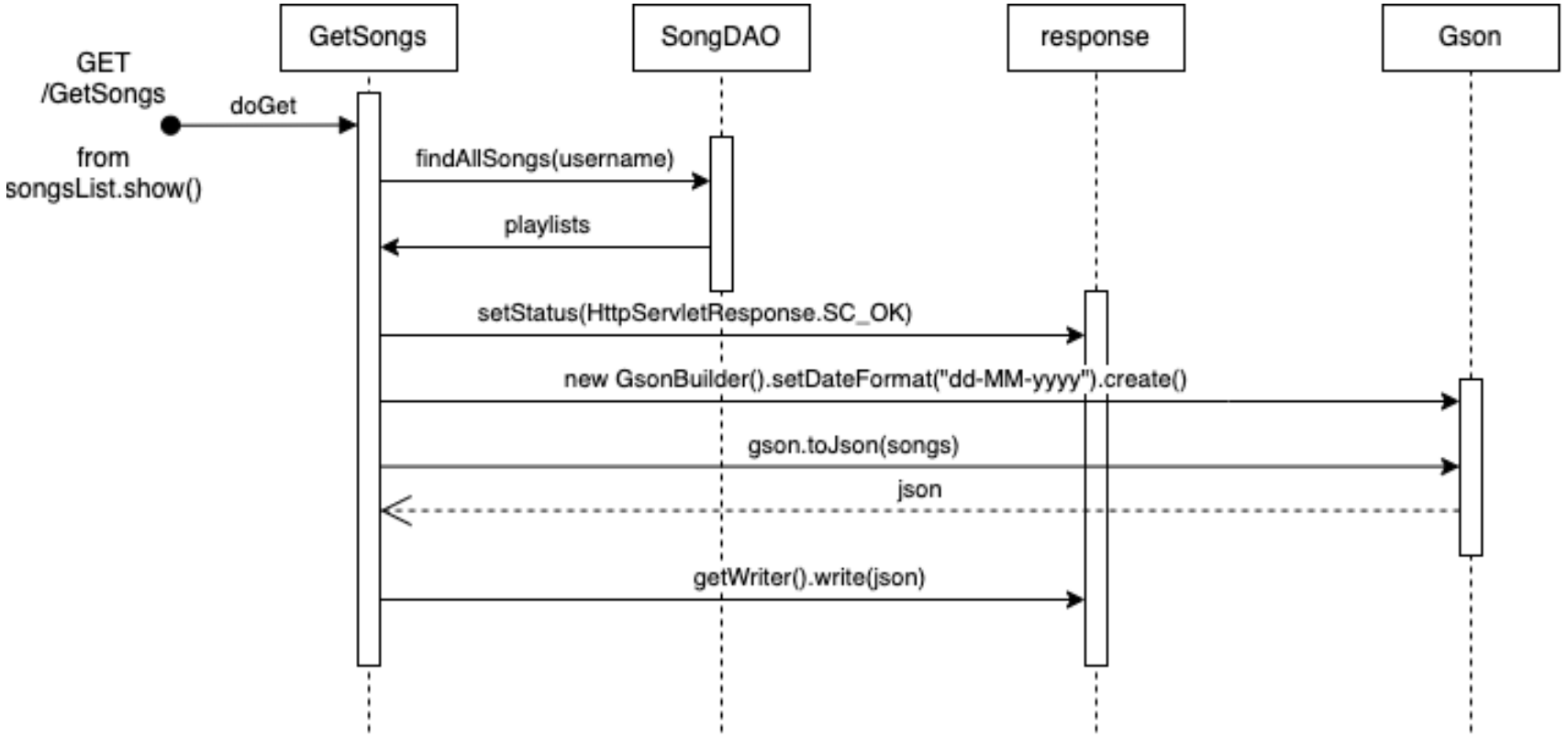
Event: create playlist



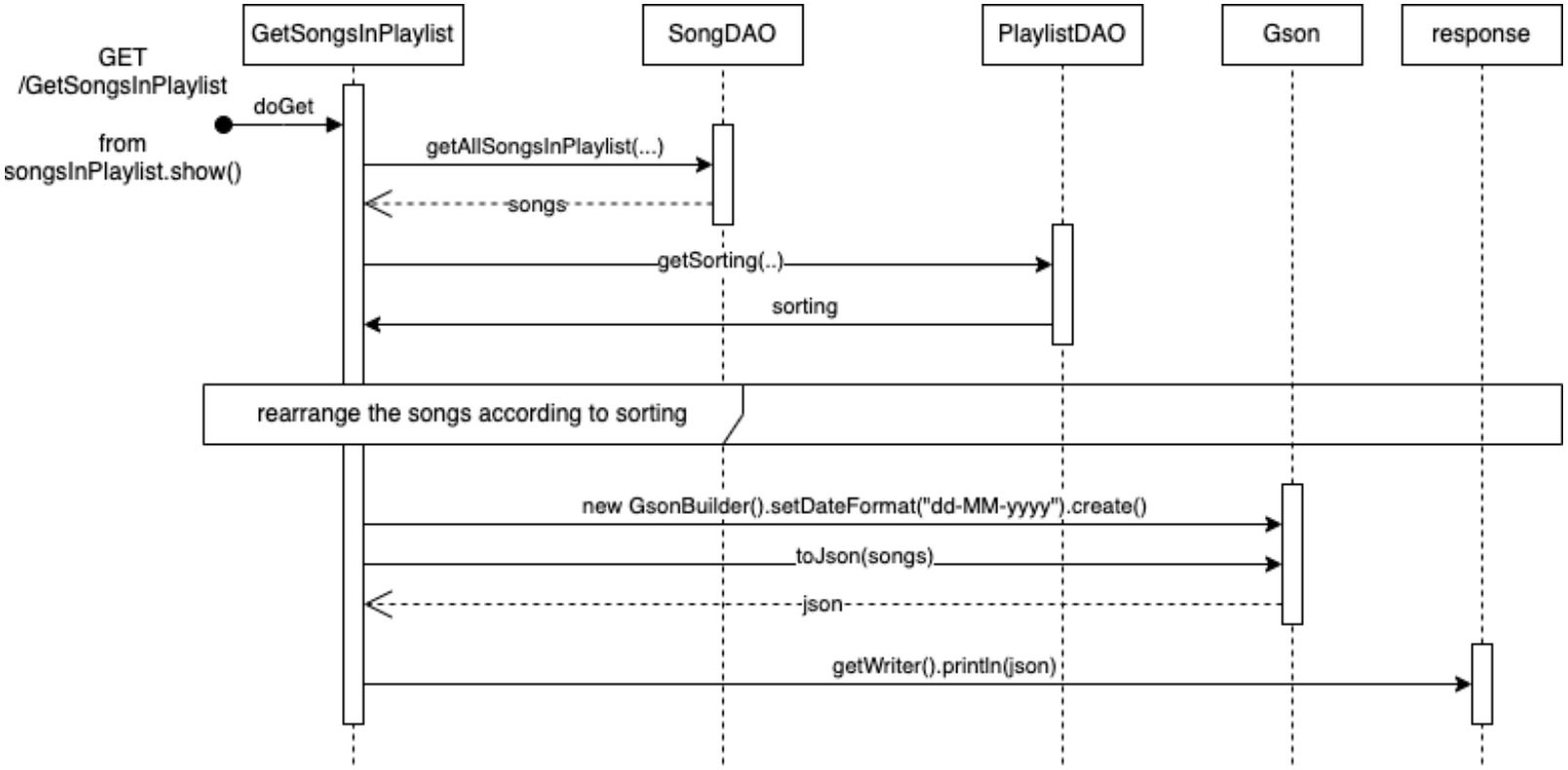
Event: get playlists



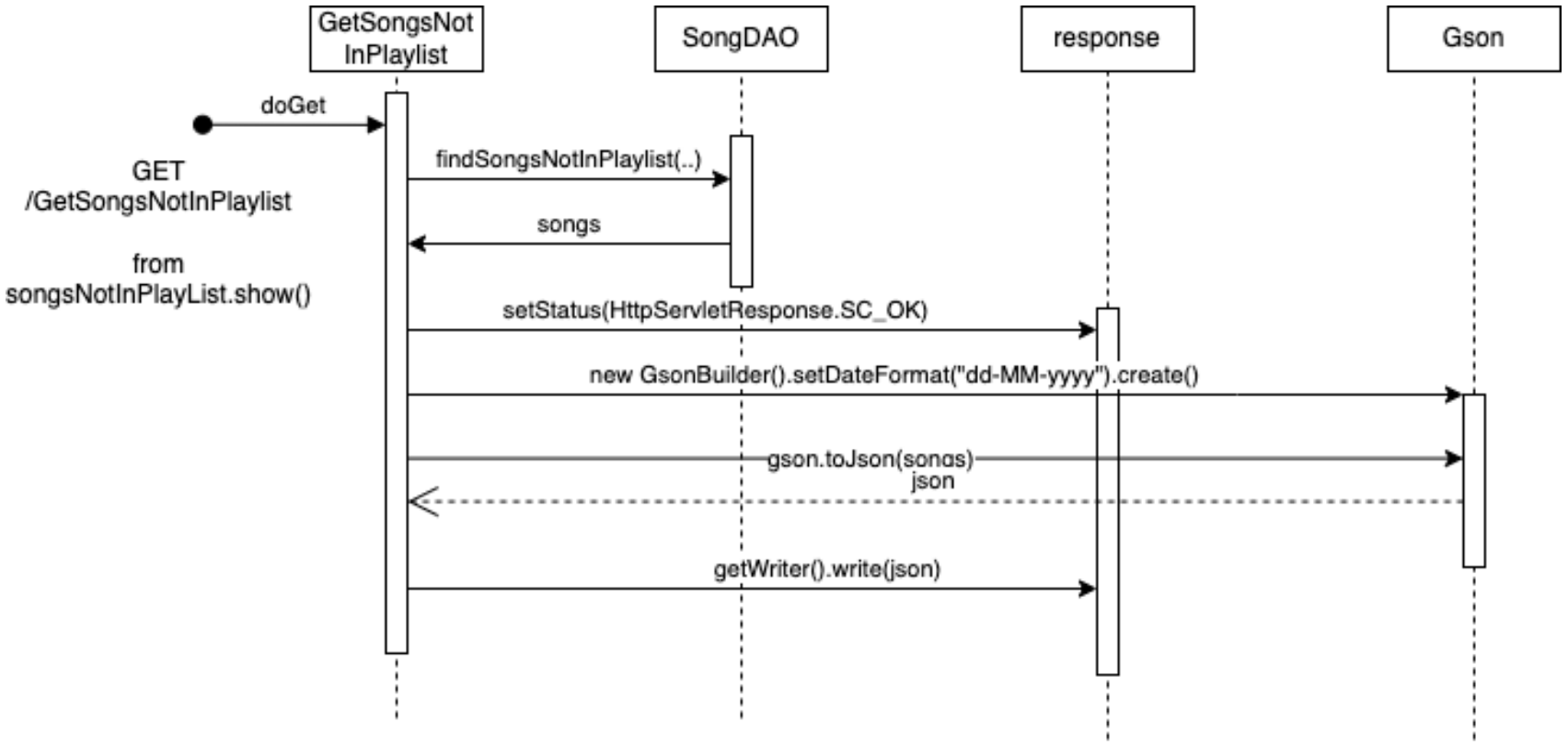
Event: get songs



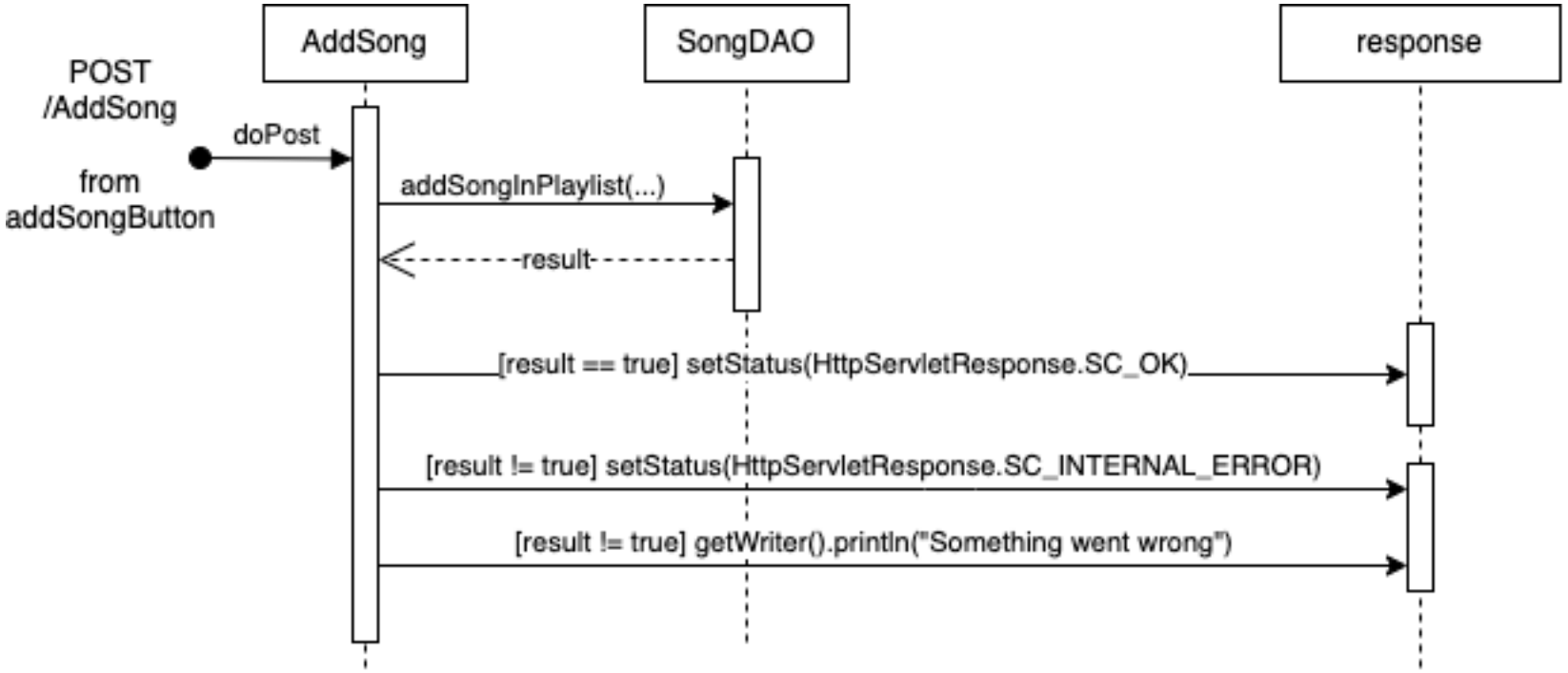
Event: get songs in playlist



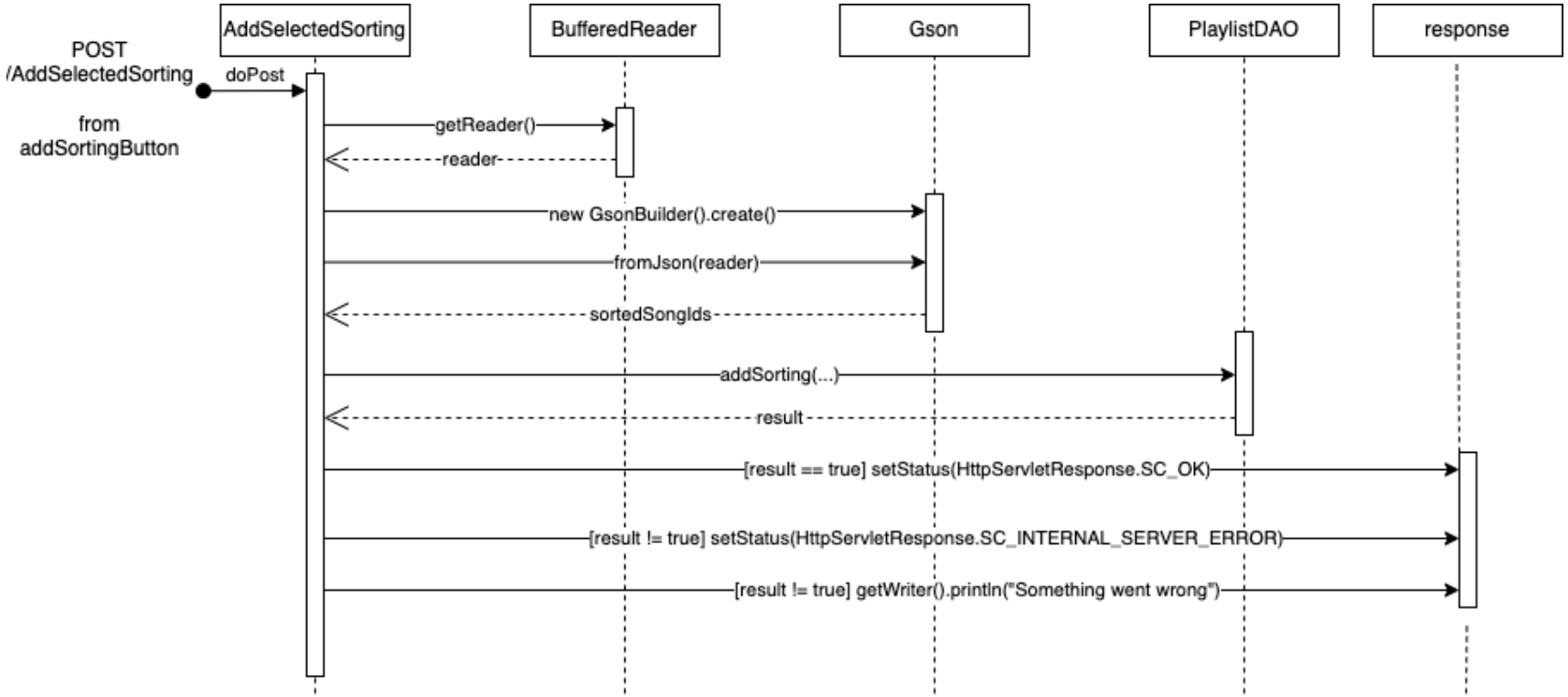
Event: get songs not in playlist



Event: add song in playlist



Event: add sorting in playlist



Event: get song details