

PlayList musicale

versione HTML pura

Progetto TIW 2024/2025

Alice Berta e Marco De Negri

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.

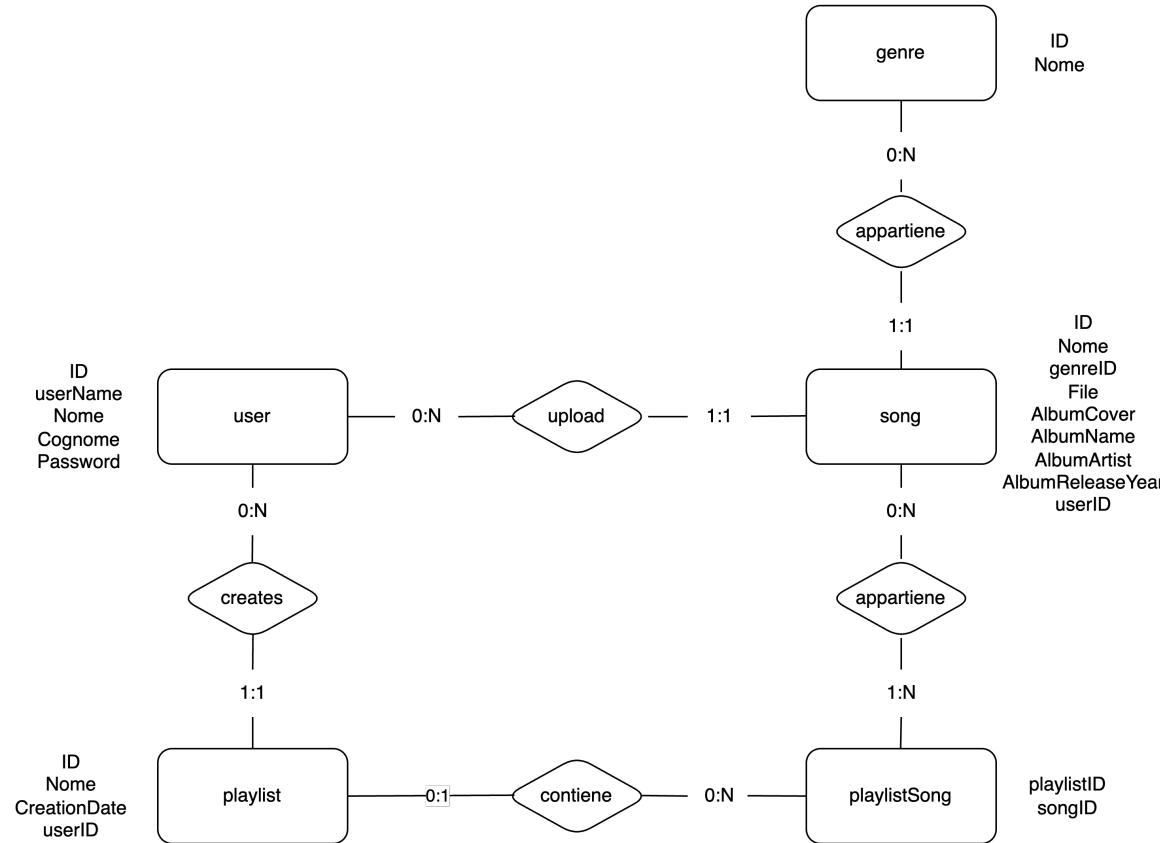
Analisi dei Dati

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 [una canzone ha associato un genere] (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.

[...] interpretazioni sulla specifica

Entità, Attributi, Relazioni

Data Base Design



Logical scheme:

USER(Id , UserName , Nome, Cognome, Password)

PLAYLIST(Id , Nome , DataCreazione , UserID)

SONG(Id , Nome, GenreID , File , AlbumCover, AlbumName, AlbumArtist, AlbumReleaseYear, userID)

GENRE(Id , Nome)

PLAYLISTSONG(PlaylistID, SongID)

Local Data Base Schema

Creazione della tabella Song

```
CREATE TABLE IF NOT EXISTS Song (
    ID INT AUTO_INCREMENT PRIMARY KEY,
    userID INT NOT NULL,
    name VARCHAR(100) NOT NULL,
    genreID INT NOT NULL,
    file VARCHAR(255) NOT NULL,
    albumCover VARCHAR(255) NOT NULL,
    albumName VARCHAR(100) NOT NULL,
    albumArtist VARCHAR(100) NOT NULL,
    albumReleaseYear INT NOT NULL,
    FOREIGN KEY (userID) REFERENCES User(ID) ON DELETE CASCADE,
    FOREIGN KEY (genreID) REFERENCES Genre(ID) ON DELETE RESTRICT
);
```

Creazione della tabella Genre

```
CREATE TABLE IF NOT EXISTS Genre (
    ID INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL UNIQUE);
```

Creazione della tabella User

```
CREATE TABLE IF NOT EXISTS User (
    ID INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    name VARCHAR(100) NOT NULL,
    surname VARCHAR(100) NOT NULL);
```

Creazione della tabella Playlist

```
CREATE TABLE IF NOT EXISTS Playlist (
    ID INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    creationDate DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    userID INT NOT NULL,
    FOREIGN KEY (userID) REFERENCES User(ID) ON DELETE CASCADE);
```

Creazione della tabella di associazione PlaylistSong

```
CREATE TABLE IF NOT EXISTS PlaylistSong (
    playlistID INT NOT NULL,
    songID INT NOT NULL,
    PRIMARY KEY (playlistID, songID),
    FOREIGN KEY (playlistID) REFERENCES Playlist(ID) ON DELETE CASCADE,
    FOREIGN KEY (songID) REFERENCES Song(ID) ON DELETE CASCADE);
```

Application requirements analysis - 1

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.

[Pagine](#), [View Component](#)

Application requirements analysis - 2

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.

Azioni, Eventi

Eventi e Azioni

Eventi

- Login
- Logout
- Submit form “carica brano”
- Submit form “crea playlist”
- Submit form “aggiungi brano a playlist corrente”
- Click playlist
- Click cella brano
- Click su successivo
- Click su precedente

Azioni

- Autenticazione
- Caricamento brano
- Creazione playlist
- Visualizzazione playlist page
- Visualizzazione player page
- Aggiungere brani a una playlist
- Navigazione tra gruppi di brani
- Reset visualizzazione al primo blocco
- Visualizzazione e riproduzione brano

Funzionalità Aggiuntive Implementate

- Abbiamo aggiunto un completo sistema di registrazione utenti con validazione dei dati, controllo unicità username e gestione errori.
- Implementate operazioni di delete per brani, playlist e account utente. Tutte includono controlli di ownership e cleanup automatico dei file associati.
- Creata una pagina dedicata per visualizzare i dati personali e permettere la cancellazione dell'account con verifica password.
- Implementato sistema di logout con invalidazione sessione e redirect alla pagina di login.
- Pattern Post-Redirect-Get implementato sistematicamente per prevenire doppio submit e gestire messaggi di feedback strutturati

Components - 1

Model Objects (Beans)

- User
- Playlist
- Song
- Genre

Views (Templates)

- index.html (Login)
- RegisterPage.html
- Home.html
- PlaylistPage.html
- PlayerPage.html
- AccountPage.html
- 404.html

Controllers (Servlets)

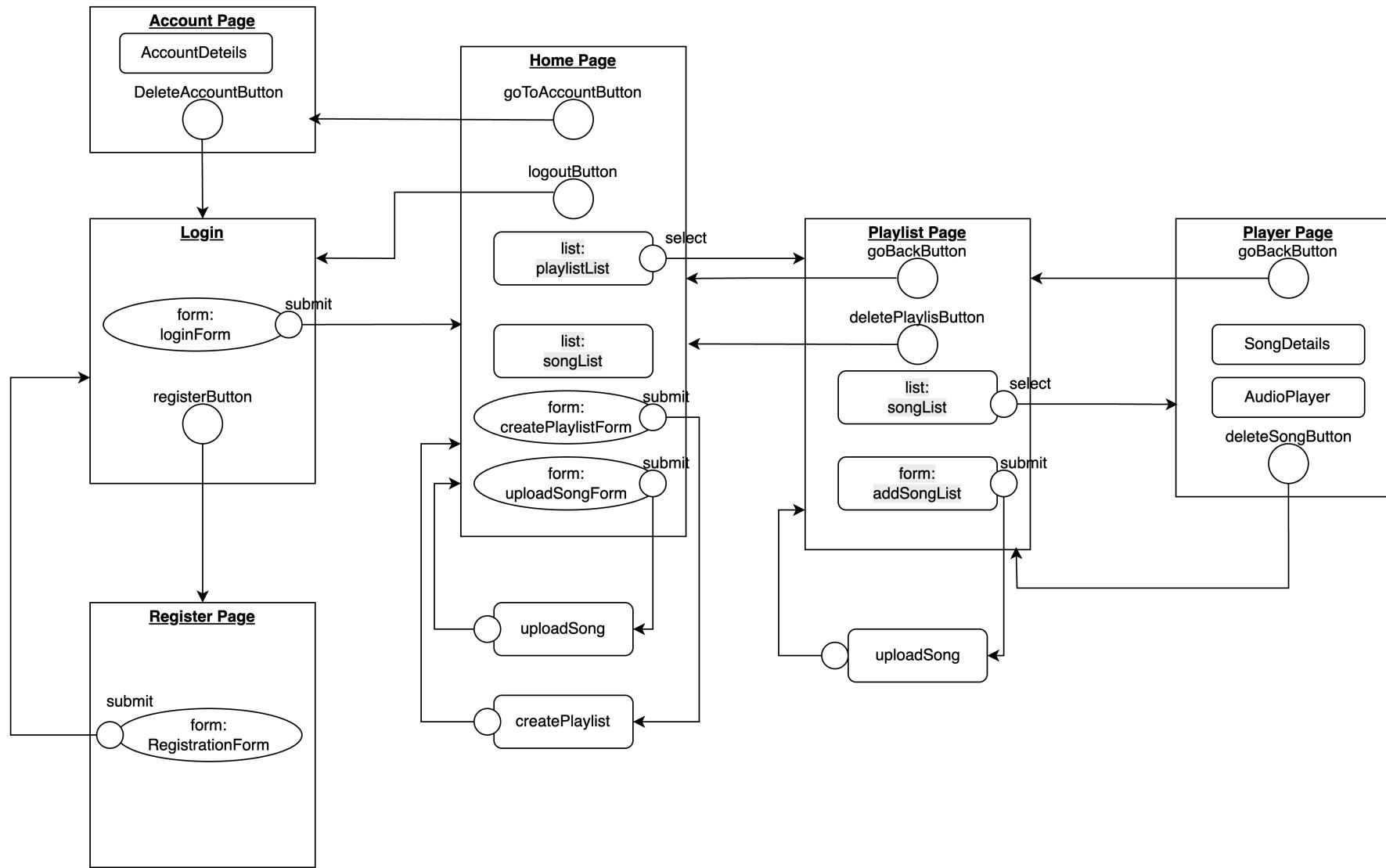
- CheckLogin
- Register
- GoToHomePage
- UploadSong
- CreatePlaylist
- GoToPlaylistPage
- AddSongsToPlaylist
- GetFile
- GoToPlayerPage
- Logout
- DeleteUser
- DeletePlaylist
- DeleteSong
- GoToAccountPage
- GoToLoginPage
- GoToRegisterPage
- GoToError404
- CssServlet

Components - 2

Data Access Objects (Classes)

- UserDAO:
 - checkCredentials(username, password)
 - deleteUser(userID)
 - insertUser(username, password, name, surname)
 - existsUsername(username)
- GenreDAO:
 - getAllGenresNames()
 - existsGenreByName(name)
 - getGenreIdByName(name)
- SongDAO:
 - SongDAO(connection)
 - existsSongByNameAndUser(name, userID)
 - existsSongWithSameData(name, albumName, artistName, albumReleaseYear, genreID, userID)
 - uploadSong(song)
 - getAllSongsByUserId(userID)
 - existAllSongsByNamesAndUser(songNames, userID)
 - existAllSongsByIdsAndUser(songIDs, userID)
 - getSongIDByNameAndUser(songName, userID)
 - getSongByNameAndUser(songName, userID)
 - getSongByIDAndUser(songId, userId)
 - deleteSong(songID, userID)
 - deletePhysicalFiles(song)
 - songBelongsToUser(songID, userID)GenreDAO
 - getAllGenresNames()
 - existsGenreByName(name)
 - getGenreIdByName(name)
- PlaylistDAO:
 - createPlaylist(name, songIDs, userID)
 - insertPlaylist(name, userID)
 - addSongToPlaylist(playlistID, songID)
 - existsPlaylistByNameAndUser(name, userID)
 - getAllPlaylistsByUserId(userID)
 - getPlaylistByIdAndUser(playlistId, userId)
 - getSongsFromPlaylist(playlistId)
 - getSongsNotInPlaylist(playlistId, userId)
 - addSongsToPlaylist(playlistId, songIDs, userId)
 - isSongInPlaylist(playlistId, songId)
 - deletePlaylist(playlistId, userId)

Application Design(IFML)



Sequence diagrams

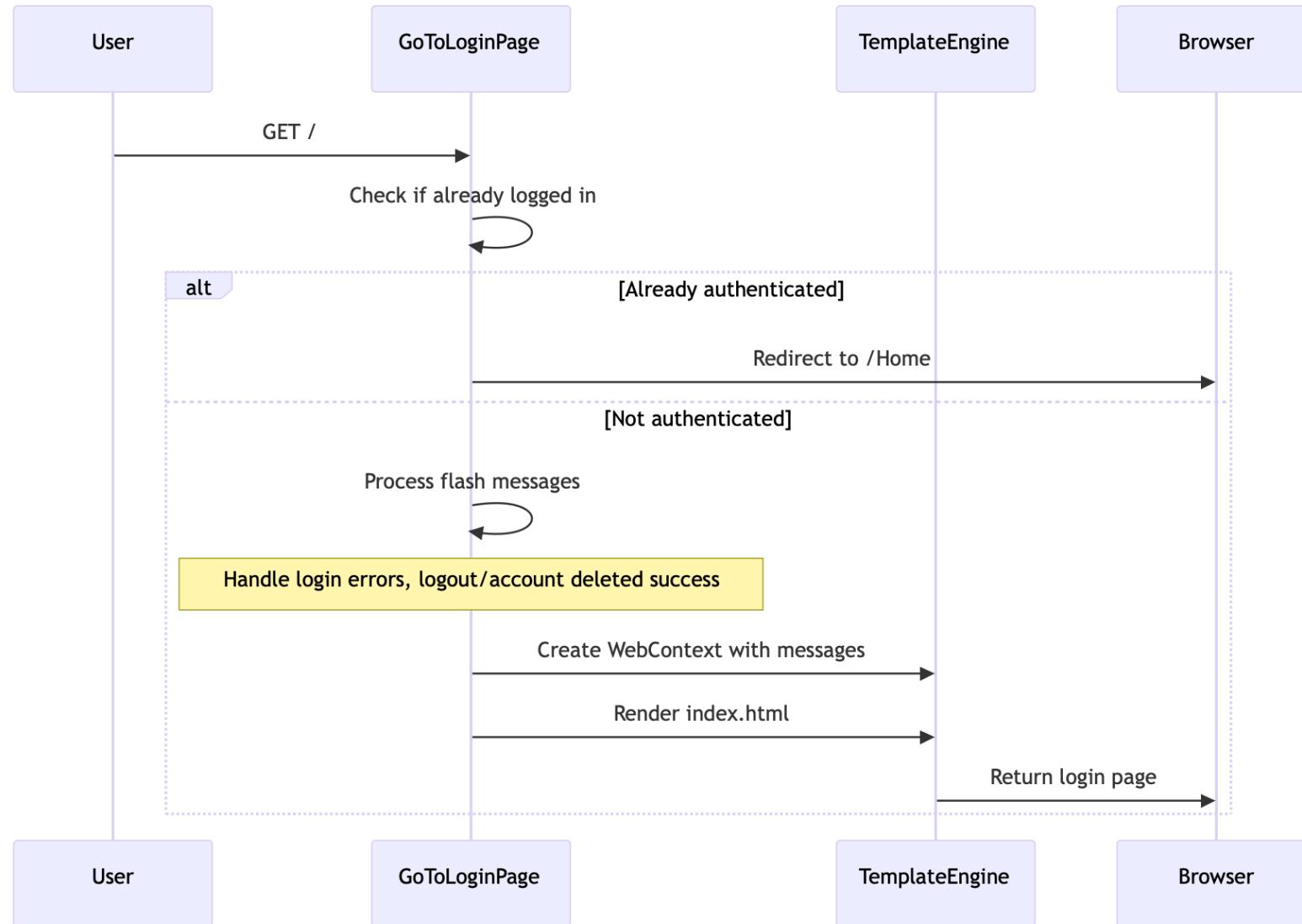
Nei sequence diagram sono state fatte alcune semplificazioni in modo da rendere più semplice la comprensione dei principali flussi dell'architettura, evitando di entrare troppo nei dettagli tecnici.

Ad esempio, la gestione dei messaggi temporanei, le validazioni complesse, la pulizia dei file temporanei e la logica dettagliata del pattern Post-Redirect-Get sono stati rappresentati in modo più sintetico.

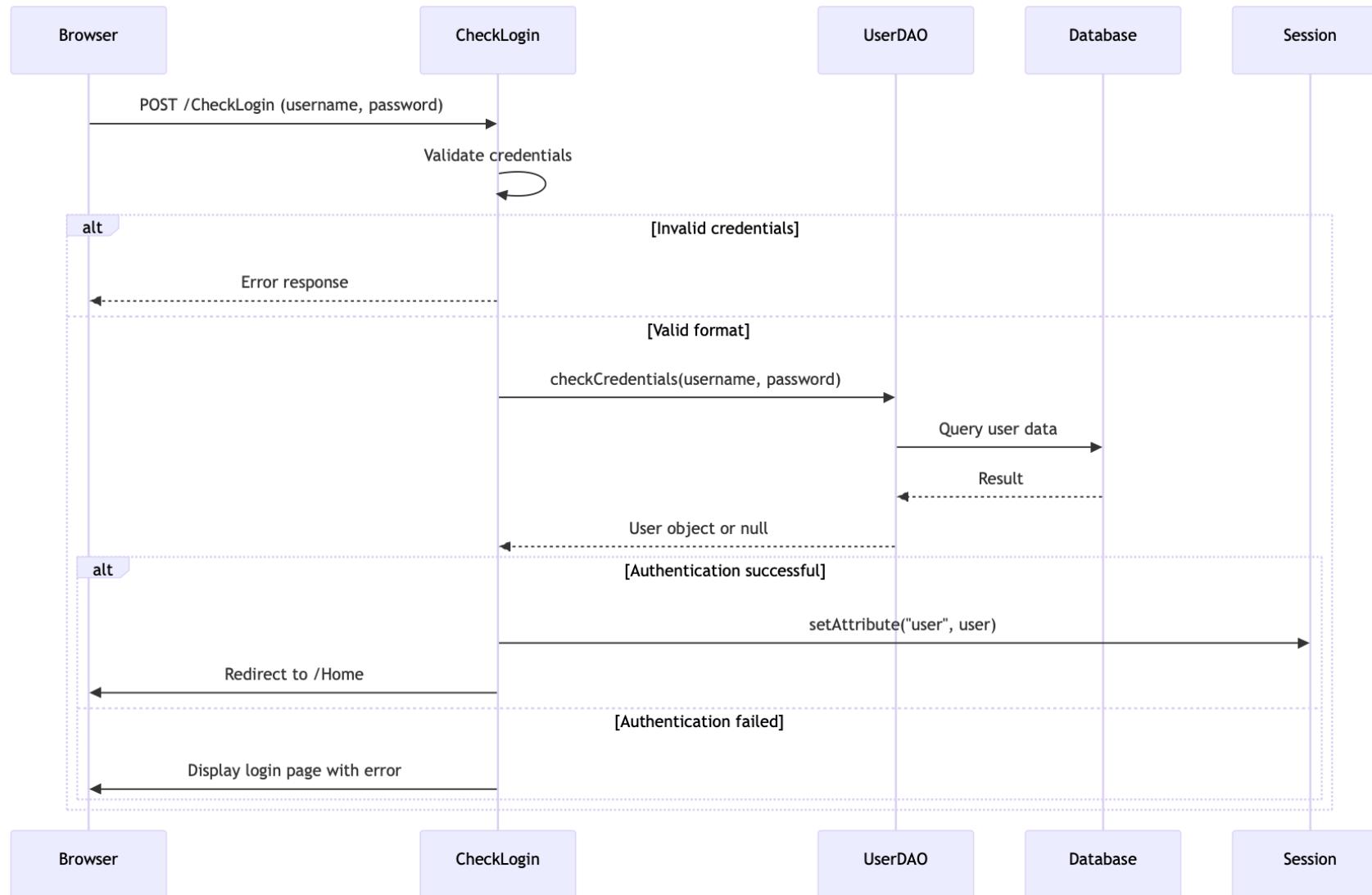
Questa scelta permette di concentrarsi sulle interazioni tra i vari componenti, rendendo i diagrammi uno strumento utile per comprendere l'architettura del sistema.

L'obiettivo è fornire una visione d'insieme chiara e accessibile.

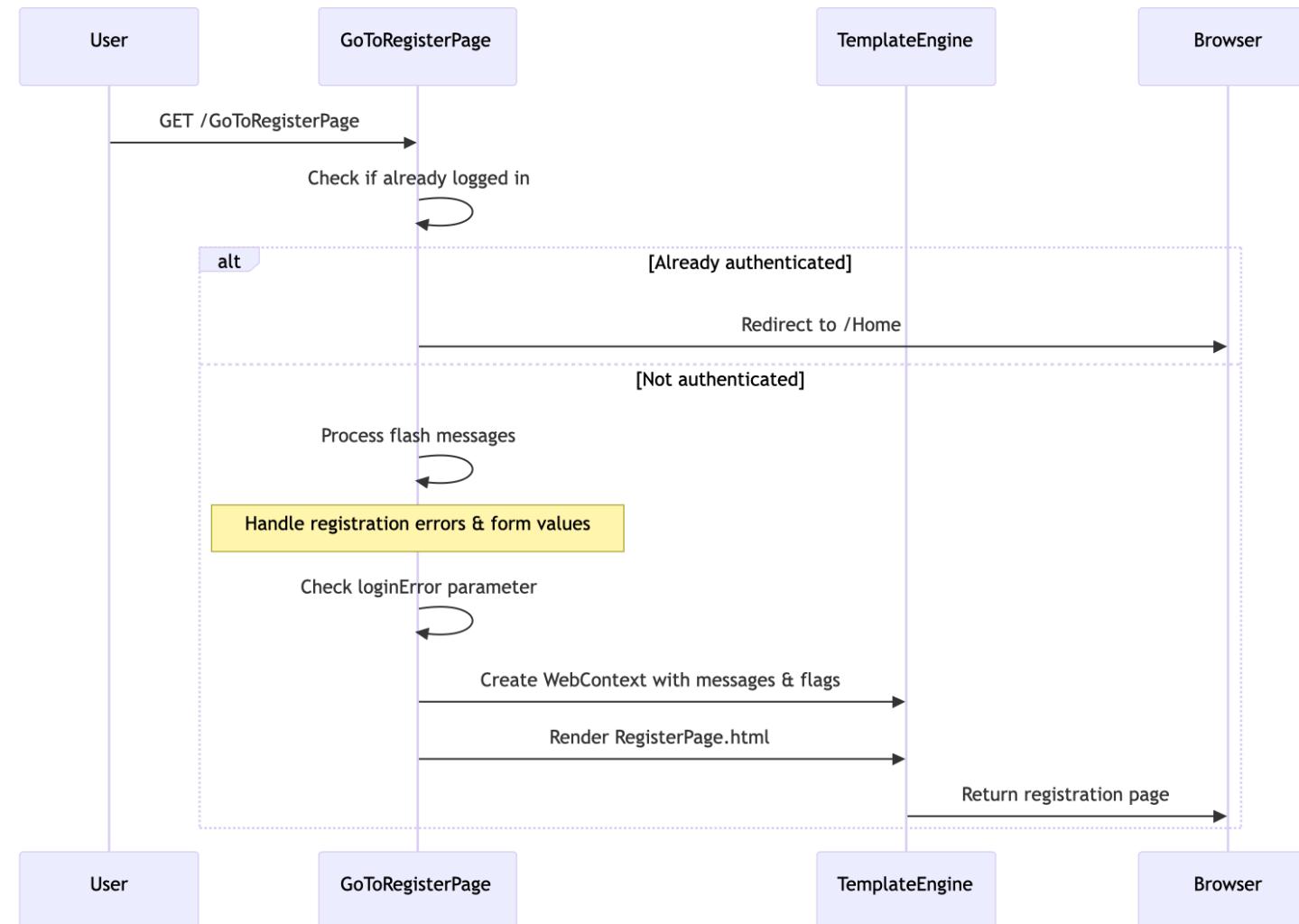
Event: GoToLoginPage



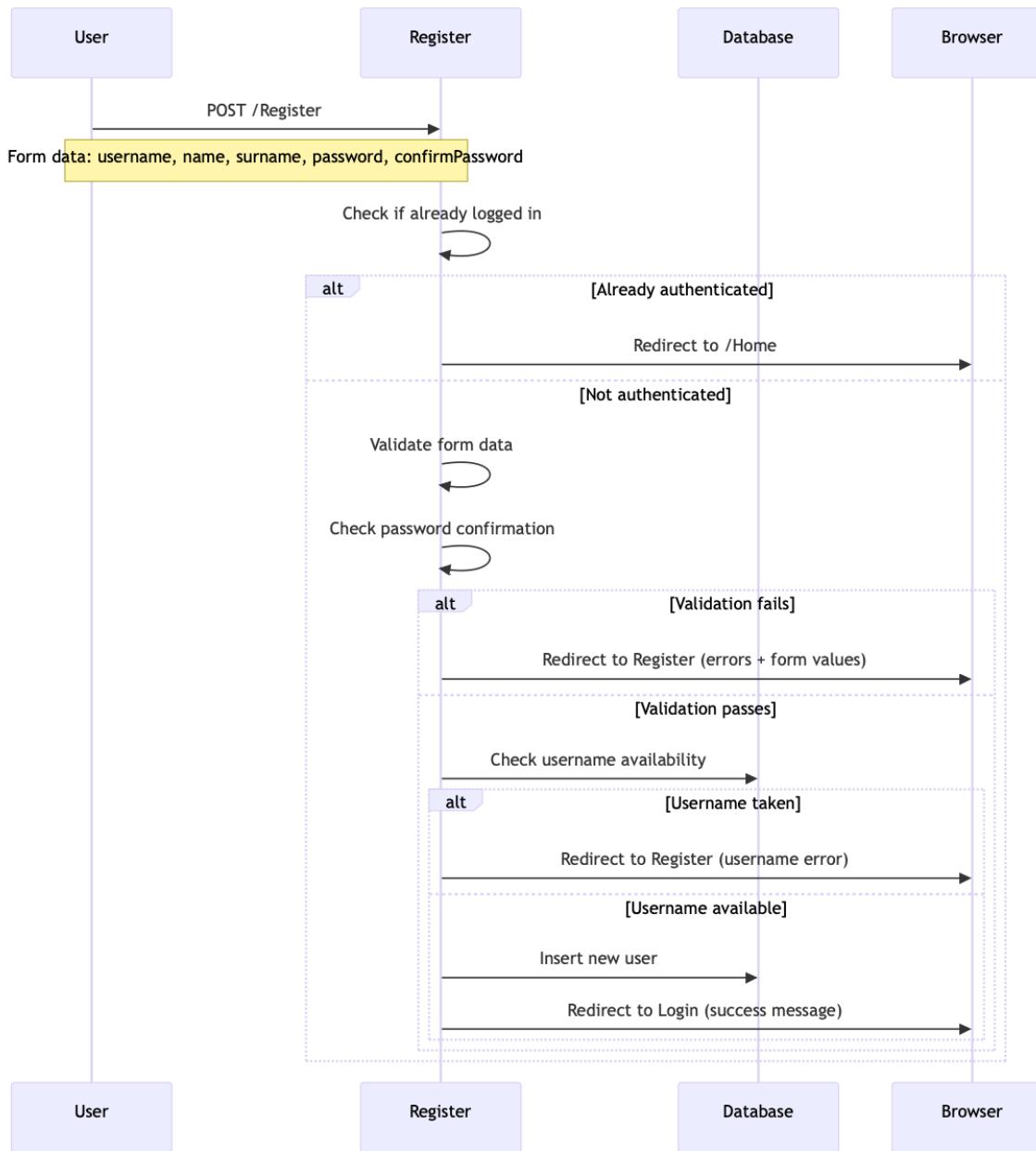
Event: CheckLogin



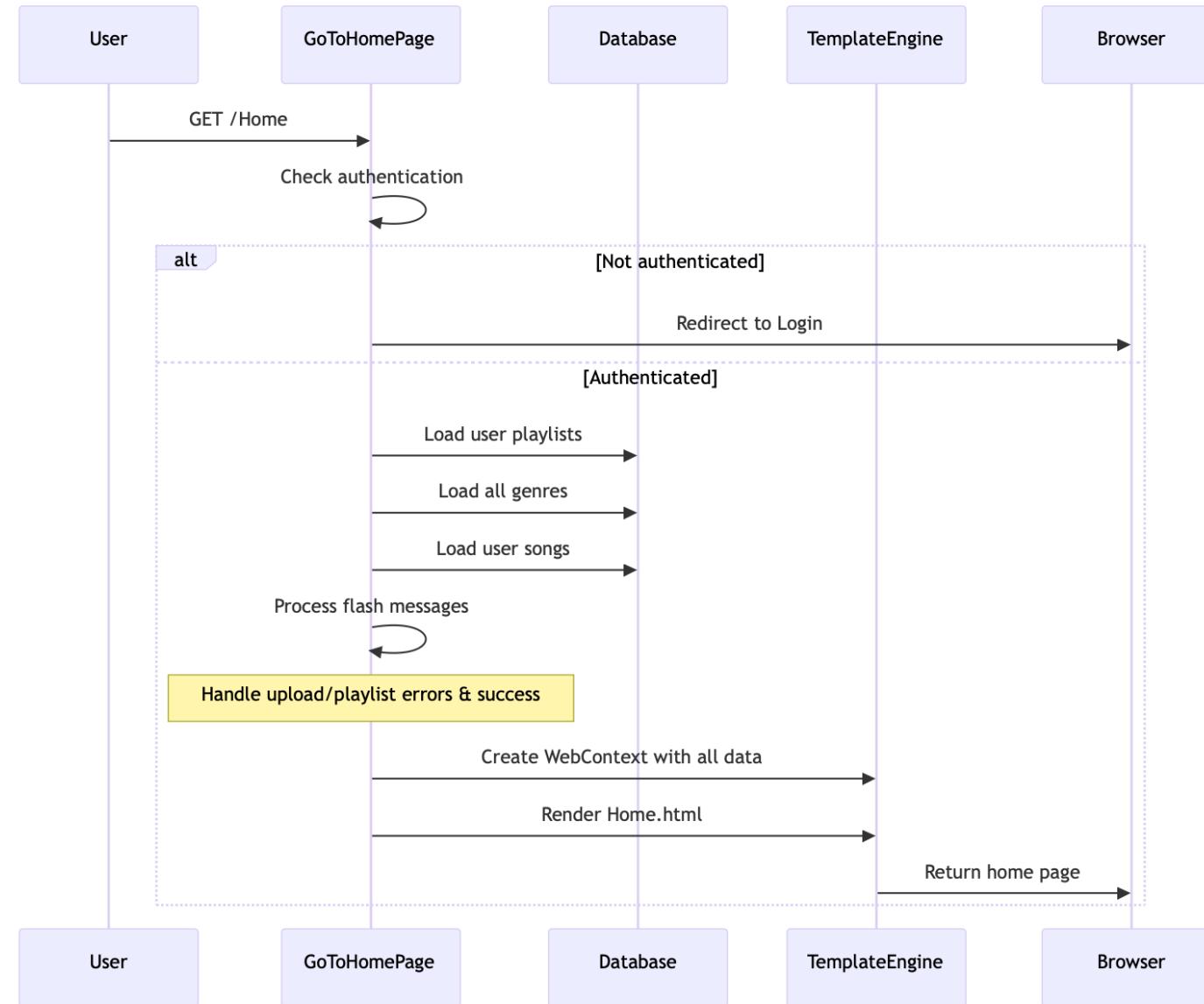
Event: GoToRegisterPage



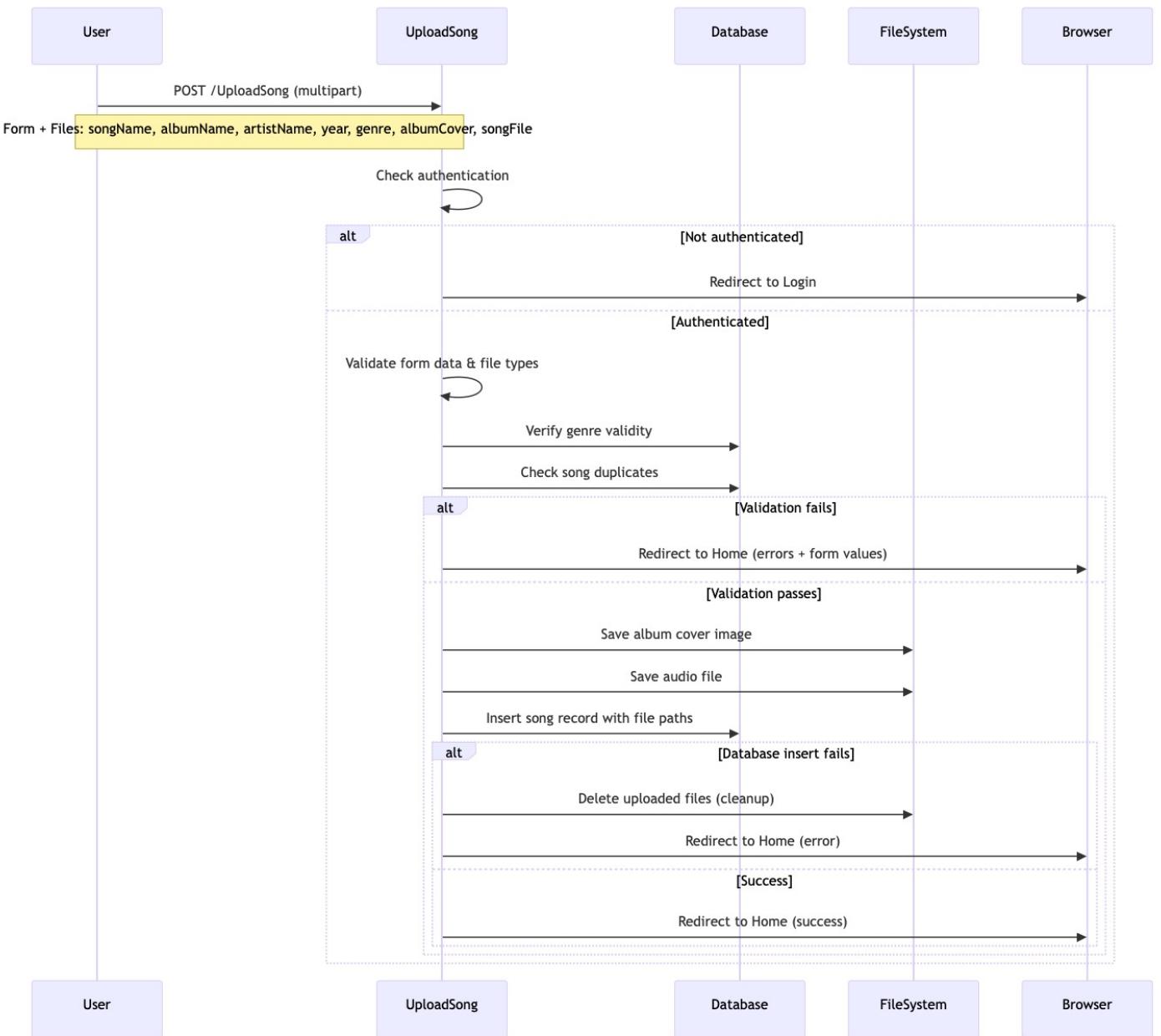
Event: Register



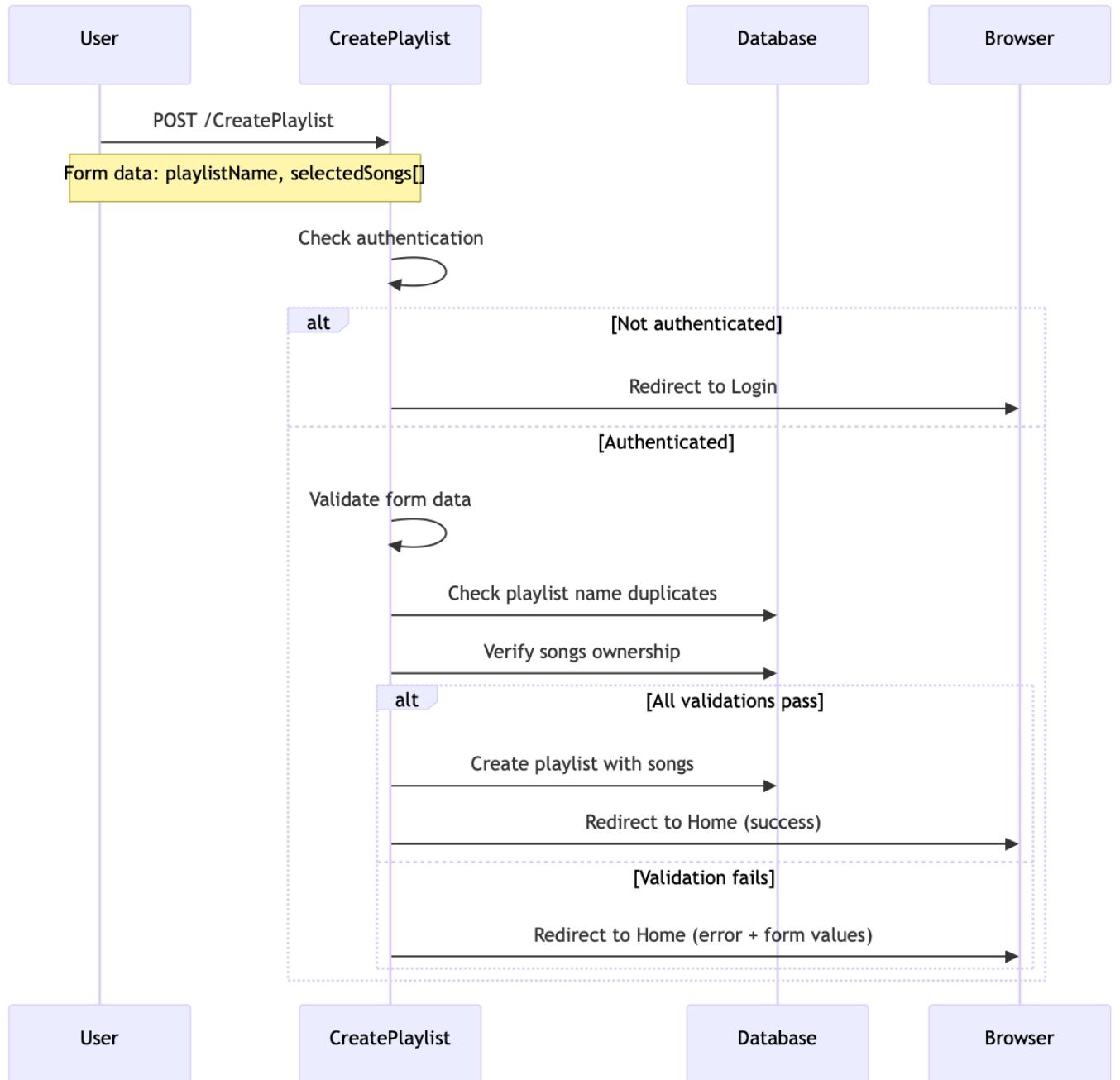
Event: GoToHomePage



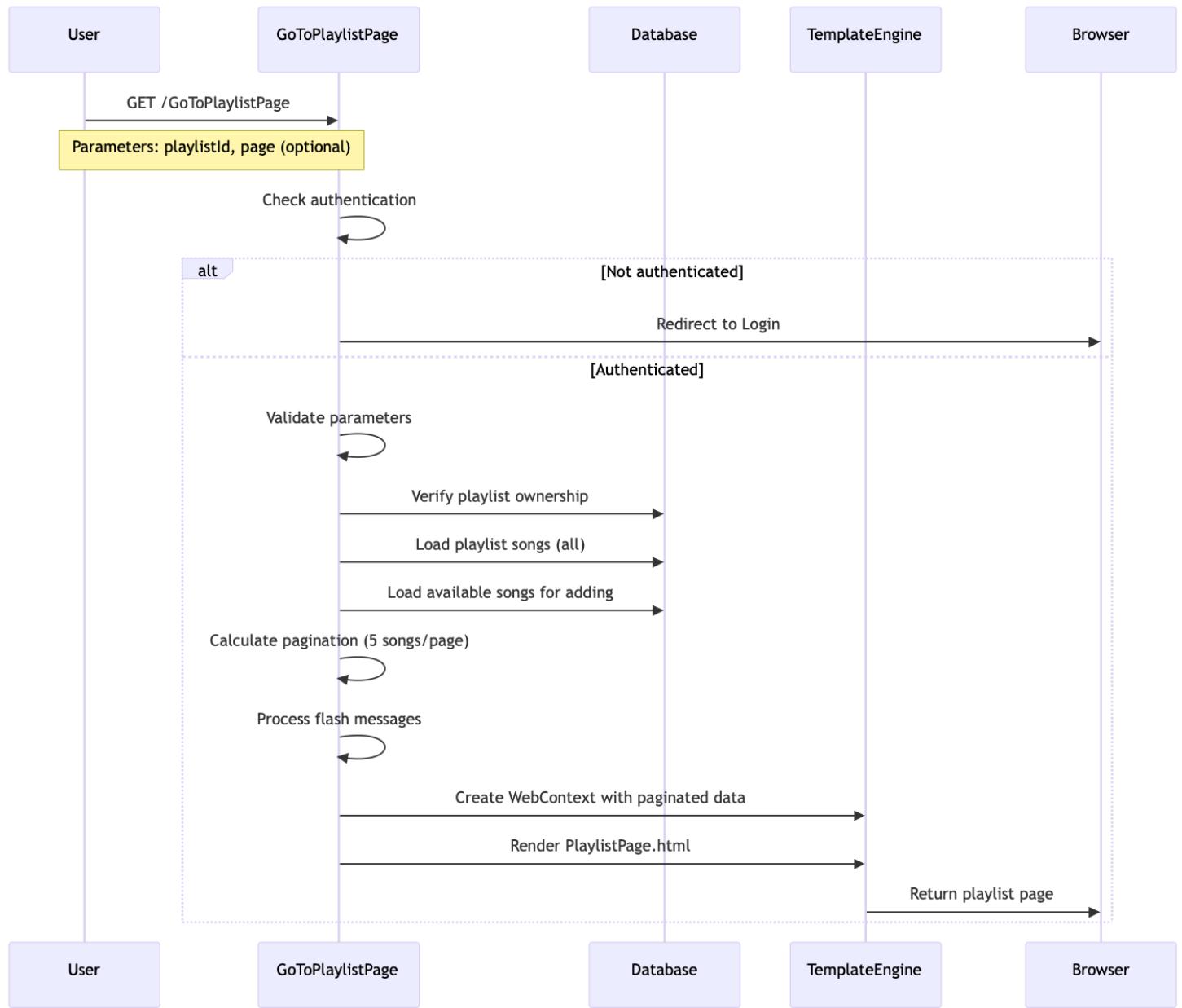
Event: UploadSong



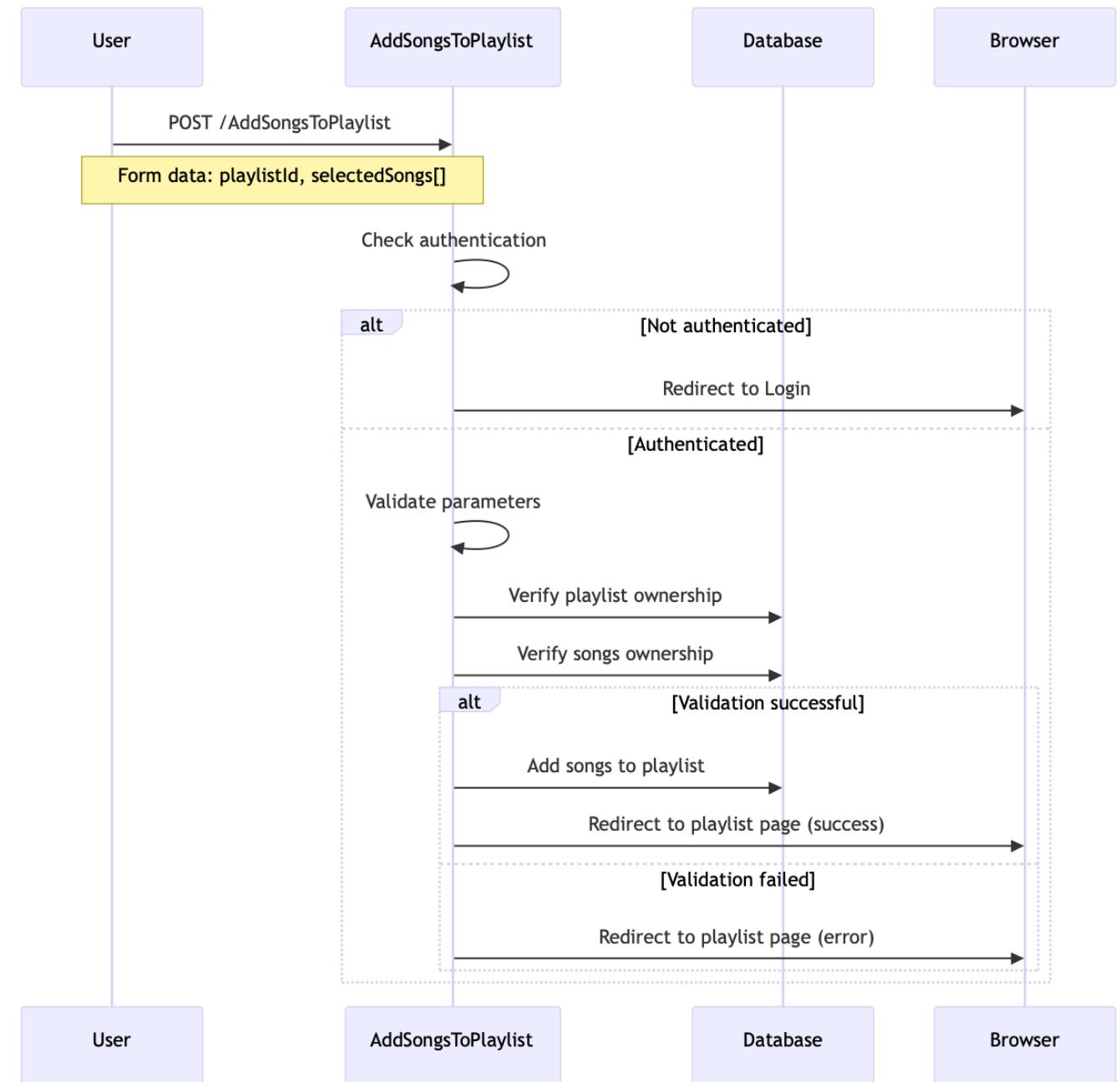
Event: CreatePlaylist



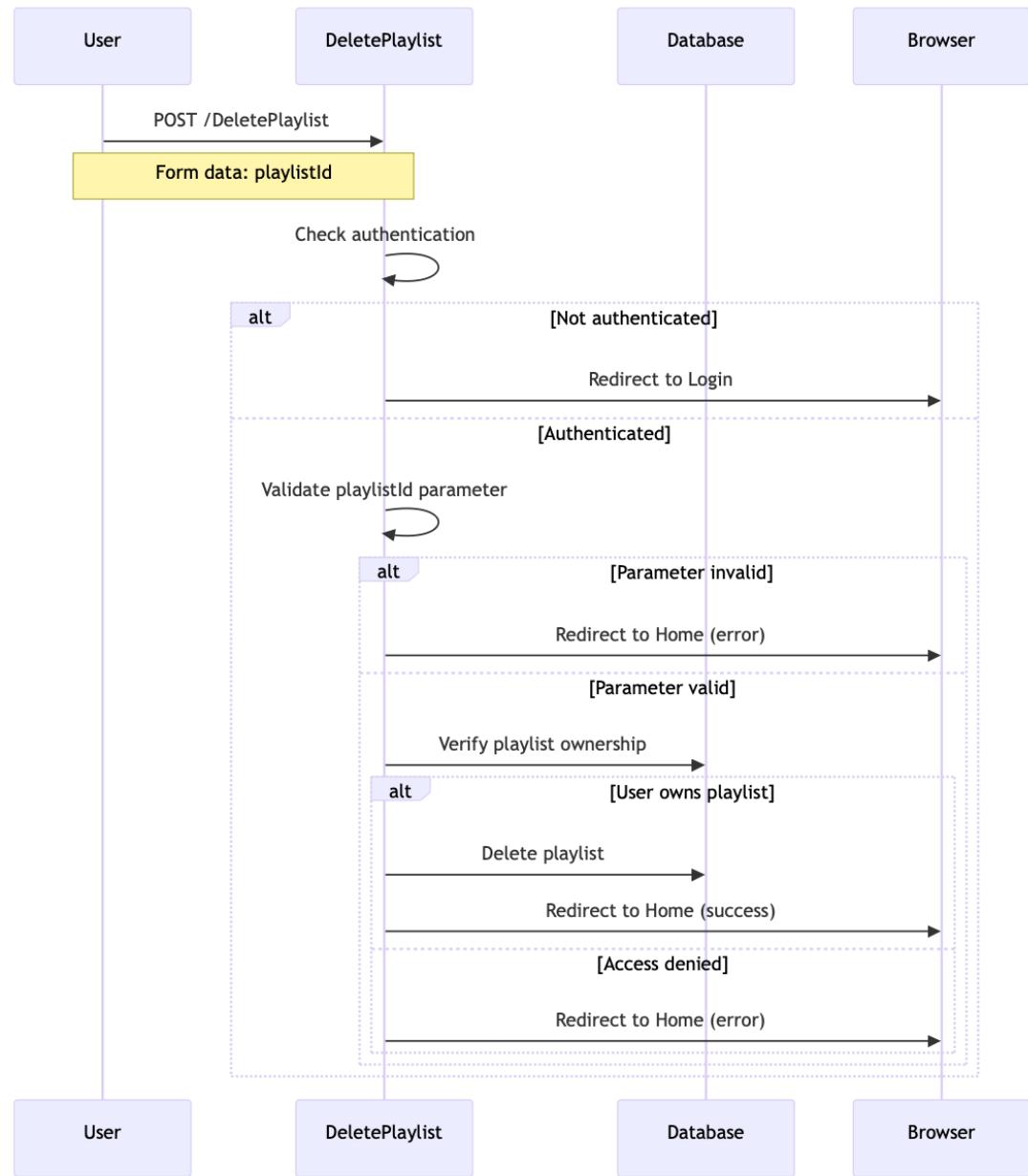
Event: GoToPlaylistPage



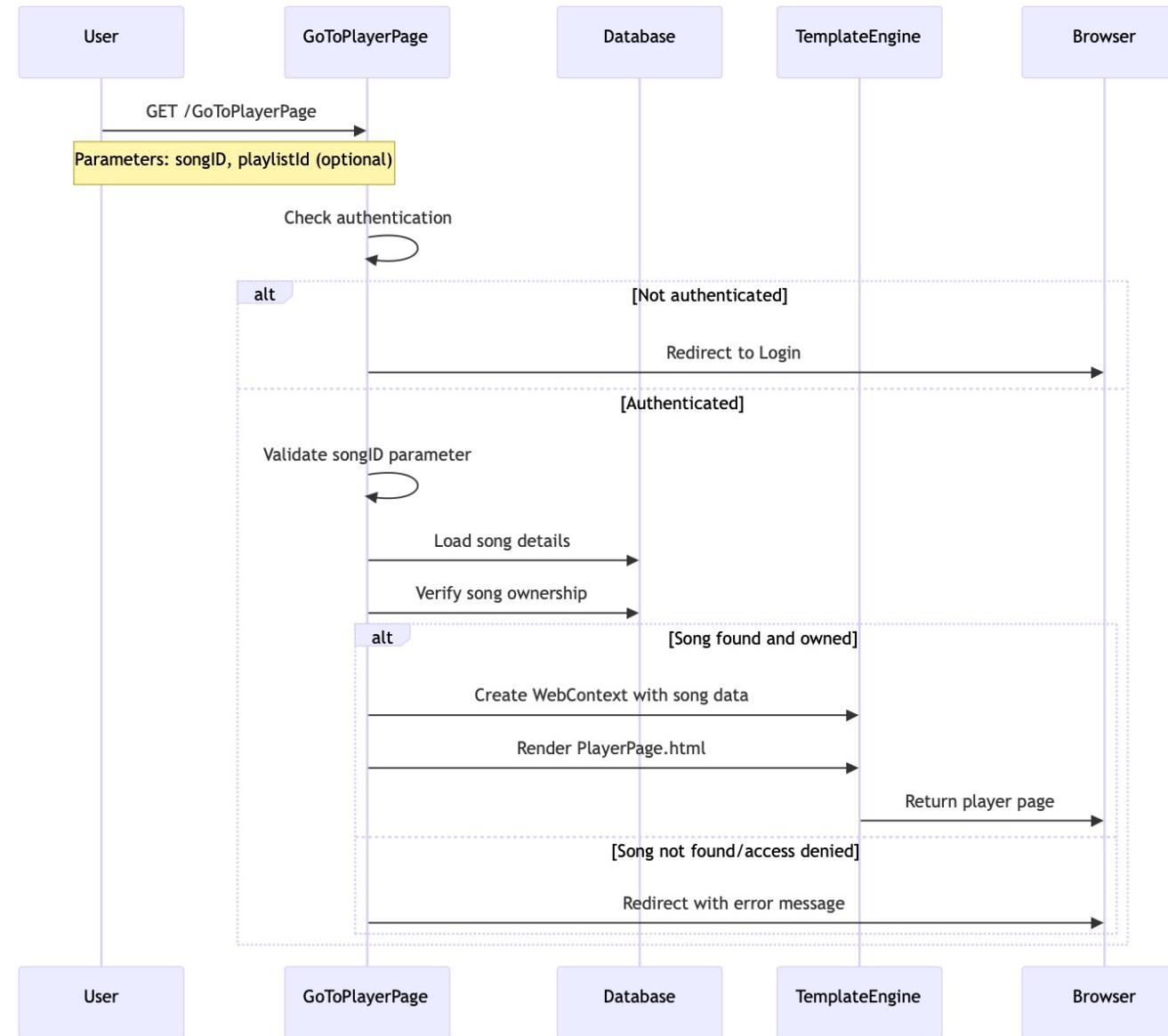
Event: AddSongToPlaylist



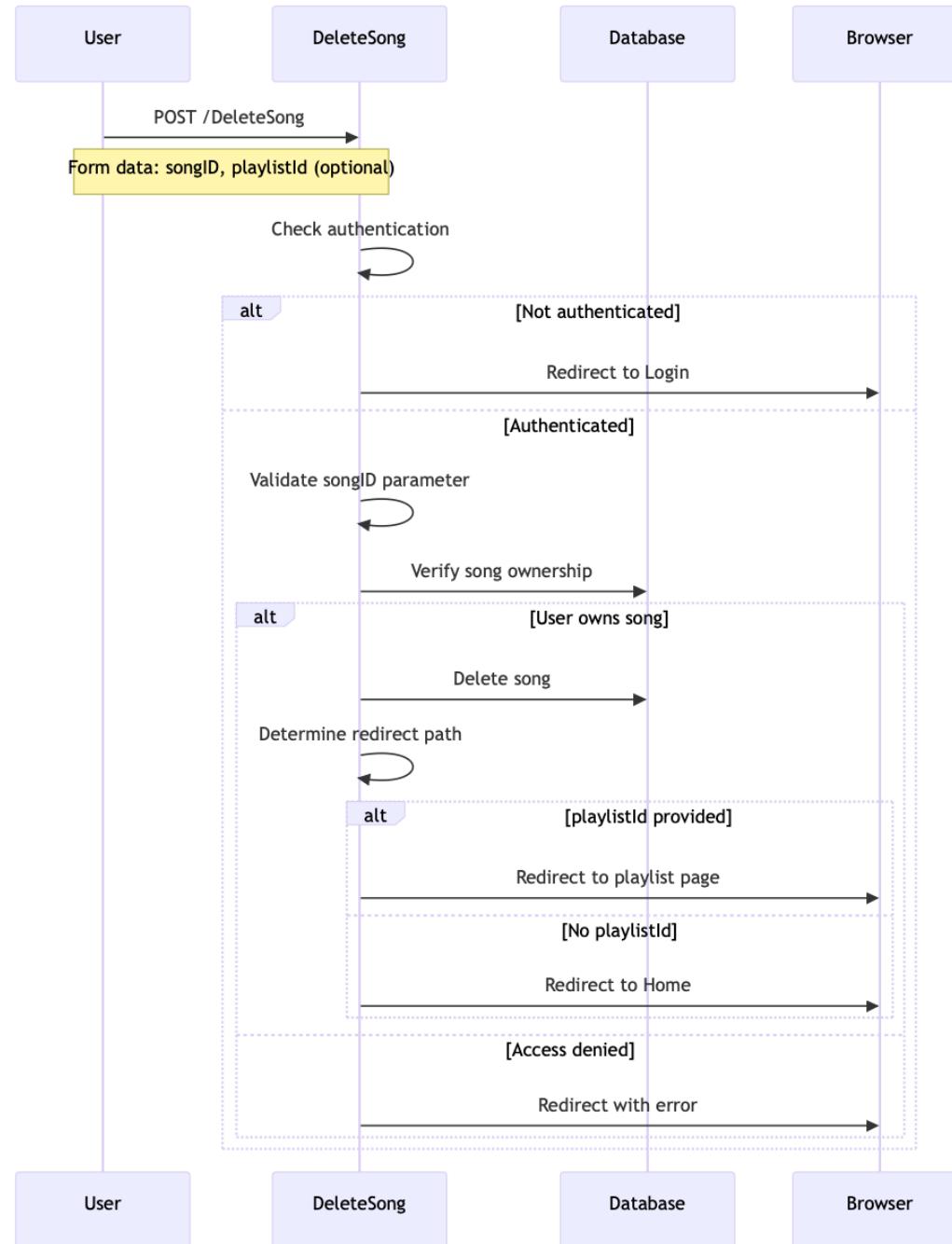
Event: DeletePlaylist



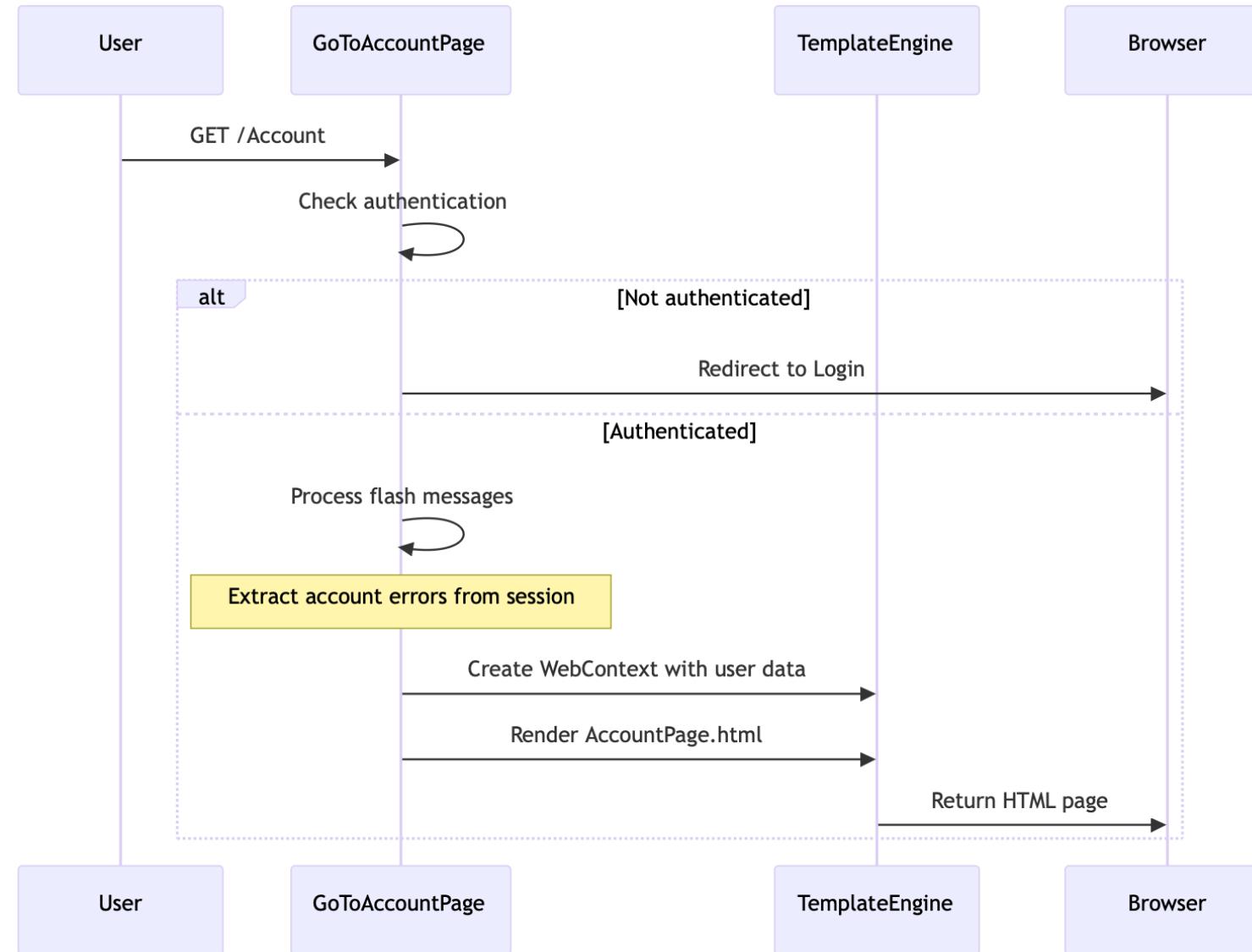
Event: GoToPlayerPage



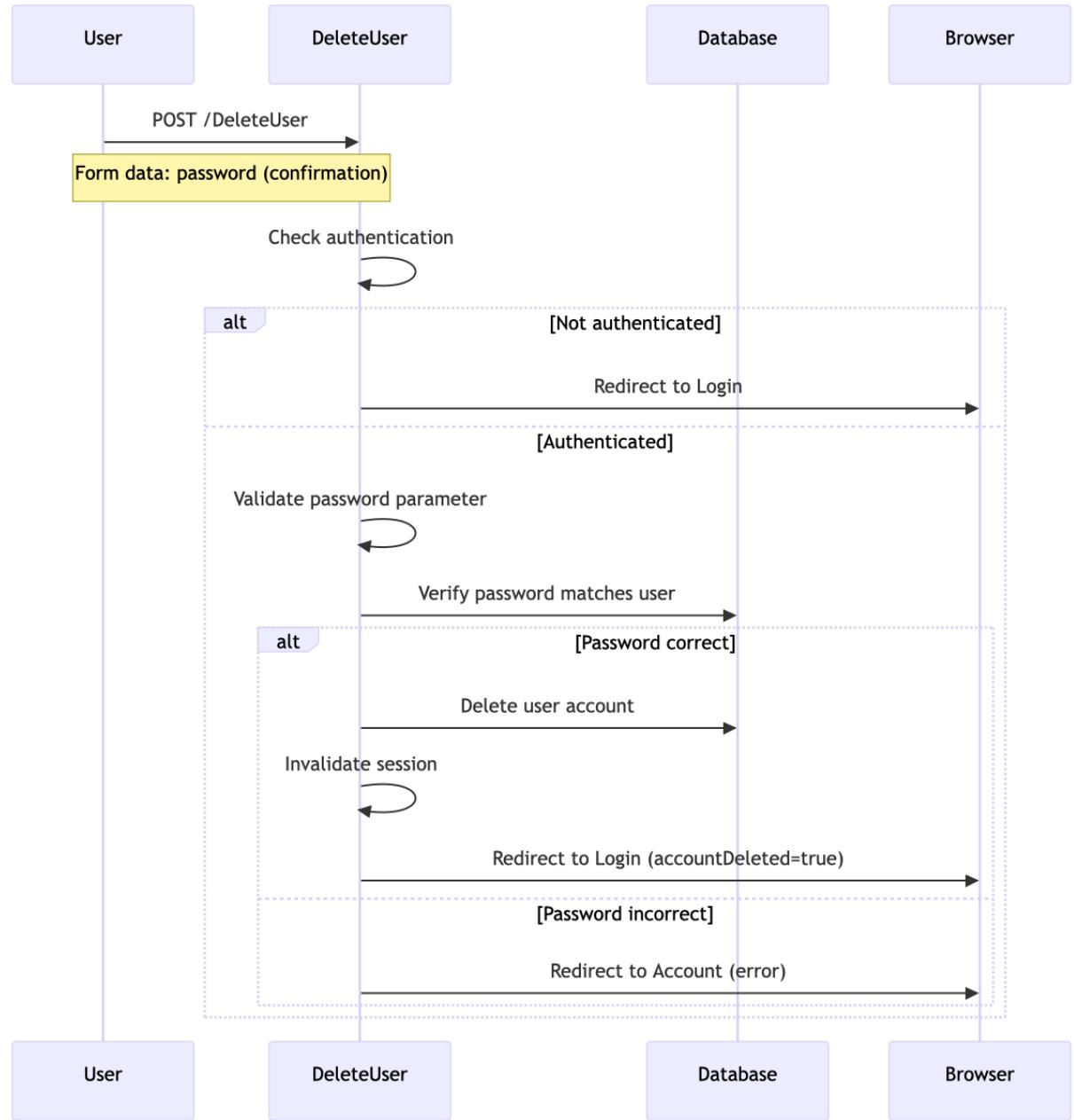
Event: DeleteSong



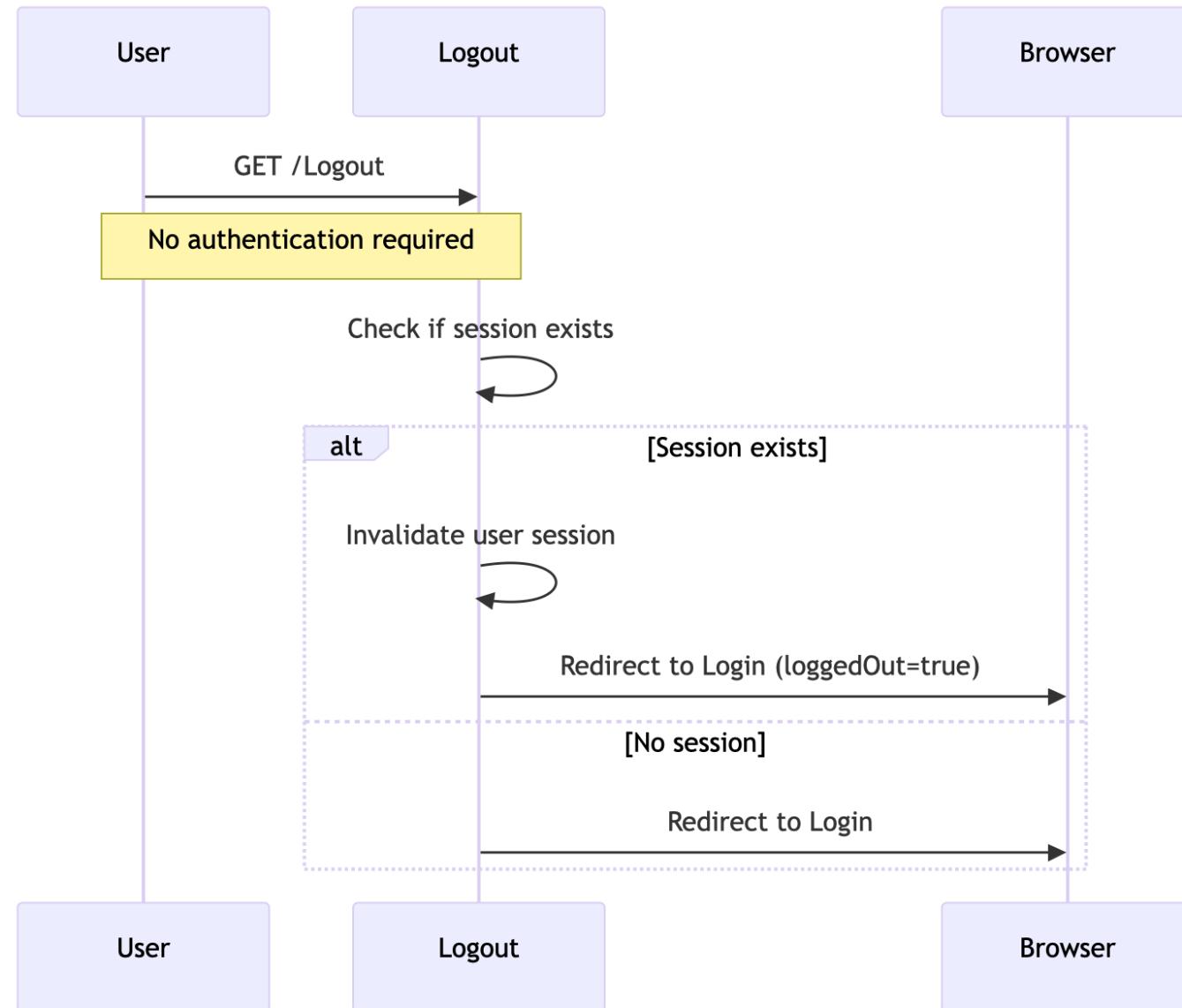
Event: GoToAccountPage



Event: DeleteUser



Event: Logout



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 ricollocarlo 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 ad una playlist con ordinamento personalizzato è inserito nell'ultima posizione.

Cambiamenti alla base di dati

Creazione della tabella di associazione PlaylistSong

```
CREATE TABLE IF NOT EXISTS PlaylistSong (
    playlistID INT NOT NULL,
    songID INT NOT NULL,
    customOrder INT, -- Added for custom playlist ordering
    PRIMARY KEY (playlistID, songID),
    FOREIGN KEY (playlistID) REFERENCES Playlist(ID) ON DELETE CASCADE,
    FOREIGN KEY (songID) REFERENCES Song(ID) ON DELETE CASCADE
);
```

È stata introdotta la colonna “customOder” per permettere l’ordinamento manuale della playlist.
La medesima base dati può essere impiegata anche nella versione in puro HTML, poiché l’informazione aggiunta non compromette in alcun modo il corretto funzionamento dell’applicazione.

Components - 1

Model Objects (Beans)

- User
- Playlist
- Song

Utilities

- ConnectionHandler
- FileStorageManager

Filters

- AuthFilter

Controllers (Servlets)

• Authentication

- CheckAuthServletRIA - GET /api/checkAuth
- LoginServletRIA - POST /api/login
- LogoutServletRIA - POST /api/logout
- RegisterServletRIA - POST /api/register
- DeleteUserServletRIA - POST /api/deleteUser

• Get content

- GenreServletRIA - GET /api/genres
- SongServletRIA - GET /api/songs, GET /api/songs/{id}
- PlaylistServletRIA - GET /api/playlists, GET /api/playlists/{id}
- FileServingServlet - GET /GetFile/*, GET /GetImage/*, GET /GetAudio/*

• Create content

- SongServletRIA - POST /api/songs (upload)
- PlaylistServletRIA - POST /api/playlists (create)
- PlaylistServletRIA - POST /api/playlists/{id}/songs (add songs)
- PlaylistServletRIA - PUT /api/playlists/{id}/order (reorder)

• Delete content

- SongServletRIA - DELETE /api/songs/{id}
- PlaylistServletRIA - DELETE /api/playlists/{id}

Components - 2

Data Access Objects (Classes)

- UserDAO:
 - checkCredentials(username, password)
 - createUser(username, password, name, surname)
 - deleteUser(userId)
 - isUsernameTaken(username)
- GenreDAO:
 - getAllGenresNames()
 - existsGenreByName(name)
 - getGenreIdByName(name)
- SongDAO:
 - uploadSong(song)
 - getAllSongsByUserId(userId)
 - getSongByIdAndUser(songId, userId)
 - deleteSong(songId, userId)
 - existsSongWithSameData(name, album, artist, year, genreId, userId)
 - existAllSongsByIdsAndUser(songIDs[], userId)
 - songBelongsToUser(songId, userId)
- PlaylistDAO:
 - createPlaylist(name, songIDs[], userId)
 - getAllPlaylistsByUserId(userId)
 - getPlaylistByIdAndUser(playlistId, userId, fetchSongs)
 - getPlaylistByNameAndUser(name, userId)
 - deletePlaylist(playlistId, userId)
 - addSongsToPlaylist(playlistId, songIDs[], userId)
 - existsPlaylistByNameAndUser(name, userId)
 - getSongsFromPlaylistOrdered(playlistId)
 - getSongsNotInPlaylist(playlistId, userId)
 - saveCustomSongOrder(playlistId, songIdsInOrder, userId)
 - getCustomSongOrder(playlistId)

Components - 3

Views

- index.html (main app container)
- login.html (authentication page)

JavaScript Utilities

- **utils.js**
 - makeCall(method, url, data, callback)
 - SessionManager
 - setUser(userData)
 - getUser()
 - clearUser()
 - hasValidUser()

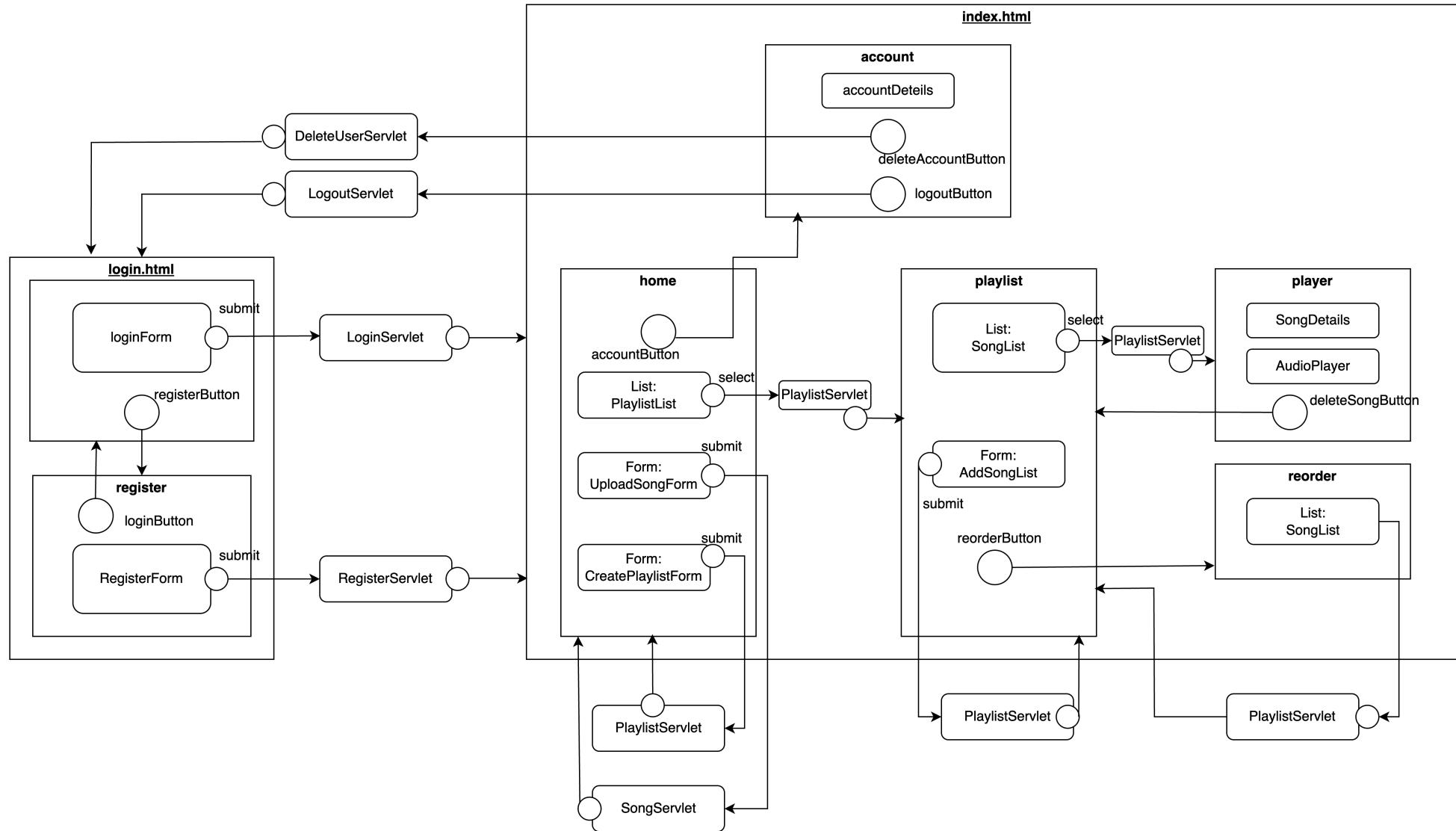
JavaScript Components

- auth.js (initLoginPage, showLoginForm, showRegisterForm, handleLogout)
- home.js (render, renderPlaylists, renderUploadSongForm, renderCreatePlaylistForm)
- playlist.js (renderDetails, renderSongGrid, renderAddSongsForm)
- player.js (render, renderSongDetails, renderAudioPlayer)
- account.js (render, handleDeleteAccount)
- reorder.js (showModal, handleSaveOrder, drag&drop handlers)

Core JavaScript

- app.js (init, fetchInitialData, reset)
- router.js (init, navigateTo)
- state.js (state management with getters/setters)

Application Design(IFML)

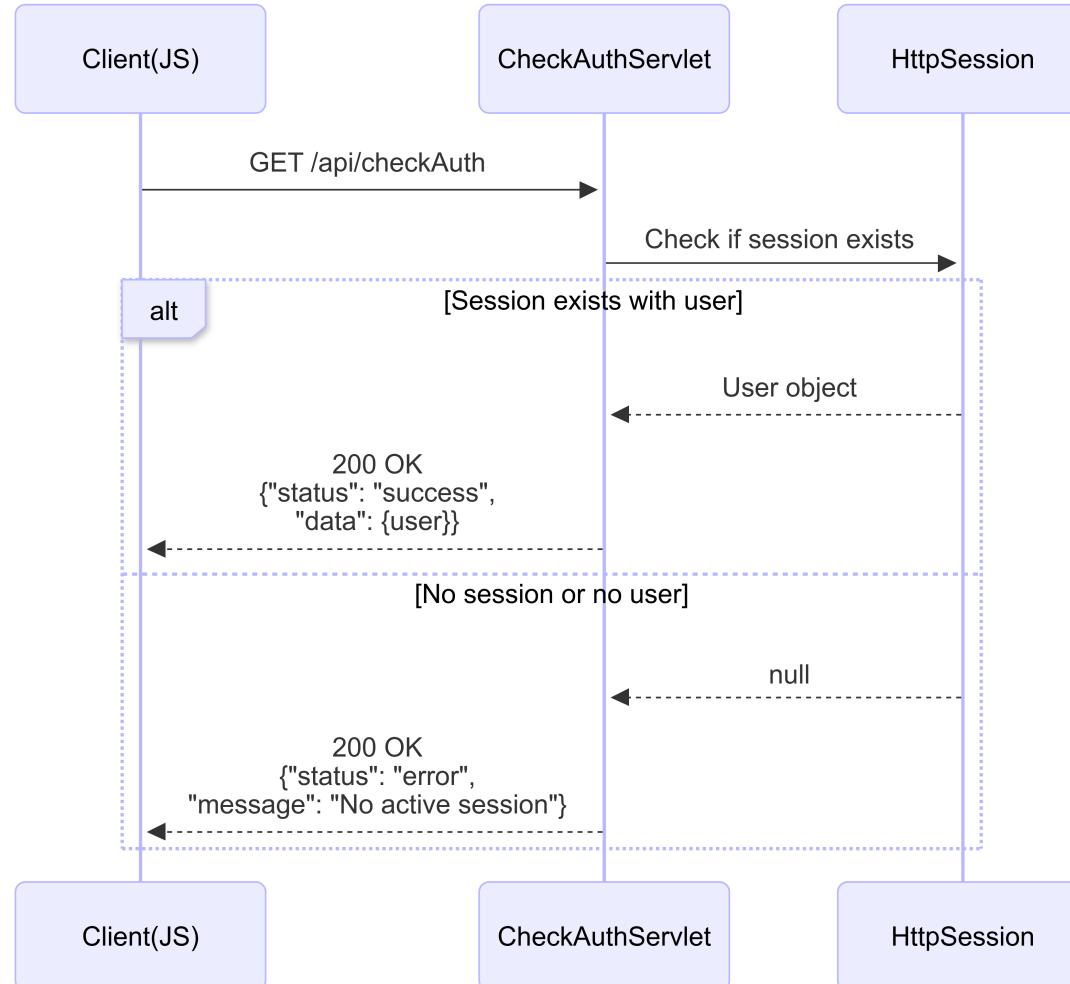


Pattern Architetturali Utilizzati

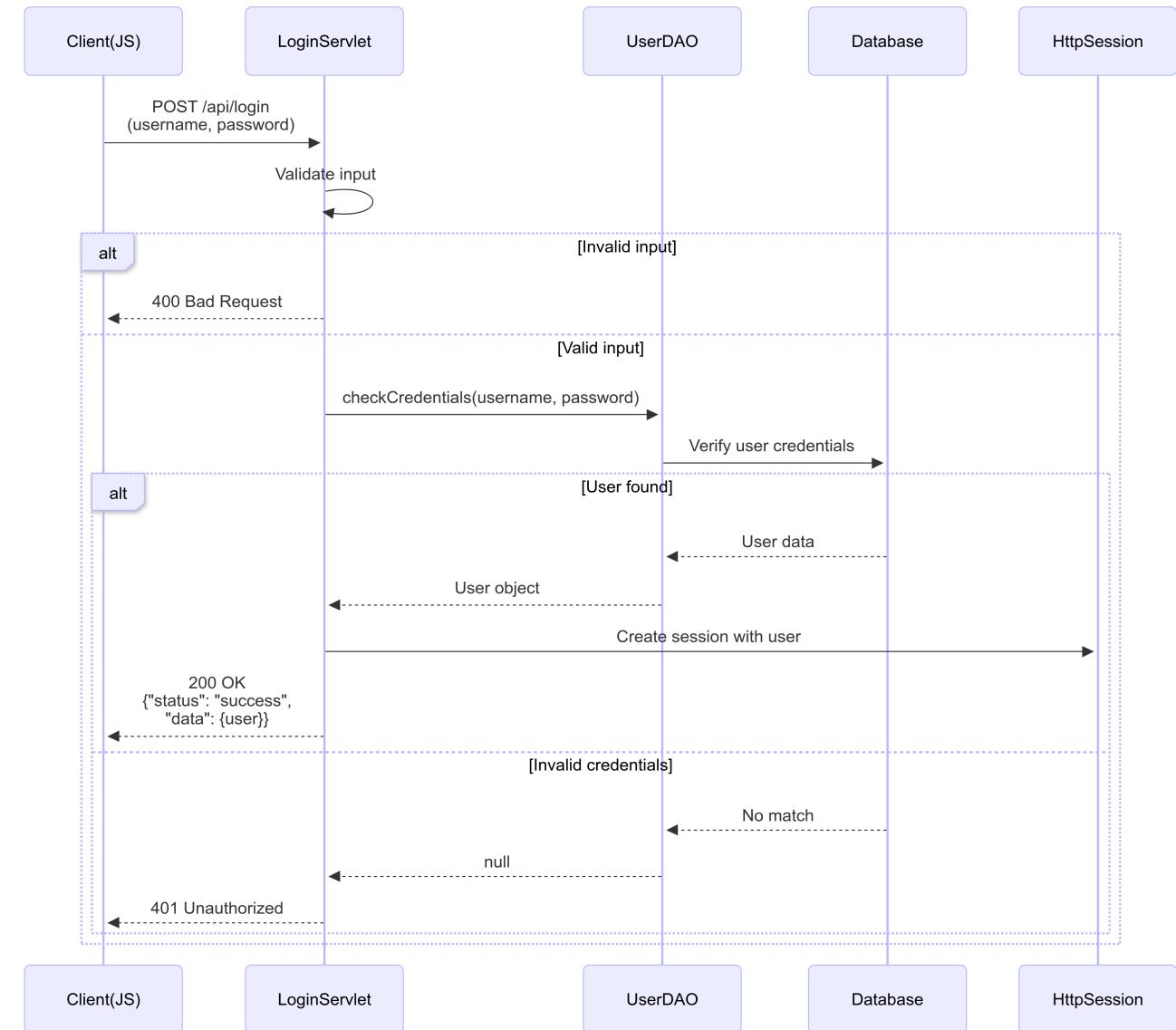
Il frontend è strutturato come una Single Page Application che utilizza un sistema di routing client-side per gestire la navigazione senza refresh della pagina.

- Lo **State Management (state.js)** implementa un pattern di gestione dello stato centralizzato con sistema di notifica basato su observer pattern. Questo sistema permette ai componenti di reagire automaticamente ai cambiamenti di stato, mantenendo sincronizzazione tra diverse parti dell'interfaccia utente.
- Mediante l'**Observer Pattern** i componenti dell'interfaccia sono “iscritti” ai cambiamenti di stato di cui sono interessati. Ad esempio, se viene modificata una playlist, i componenti interessati vengono notificati in automatico e si aggiornano senza bisogno di interventi manuali.
- Il **Router Pattern (router.js)** implementa un sistema di navigazione che mappa route specifiche a componenti di renderizzazione. Questo approccio semplifica la gestione della navigazione dell'applicazione mantenendo lo stato e permettendo transizioni fluide tra diverse viste dell'applicazione.

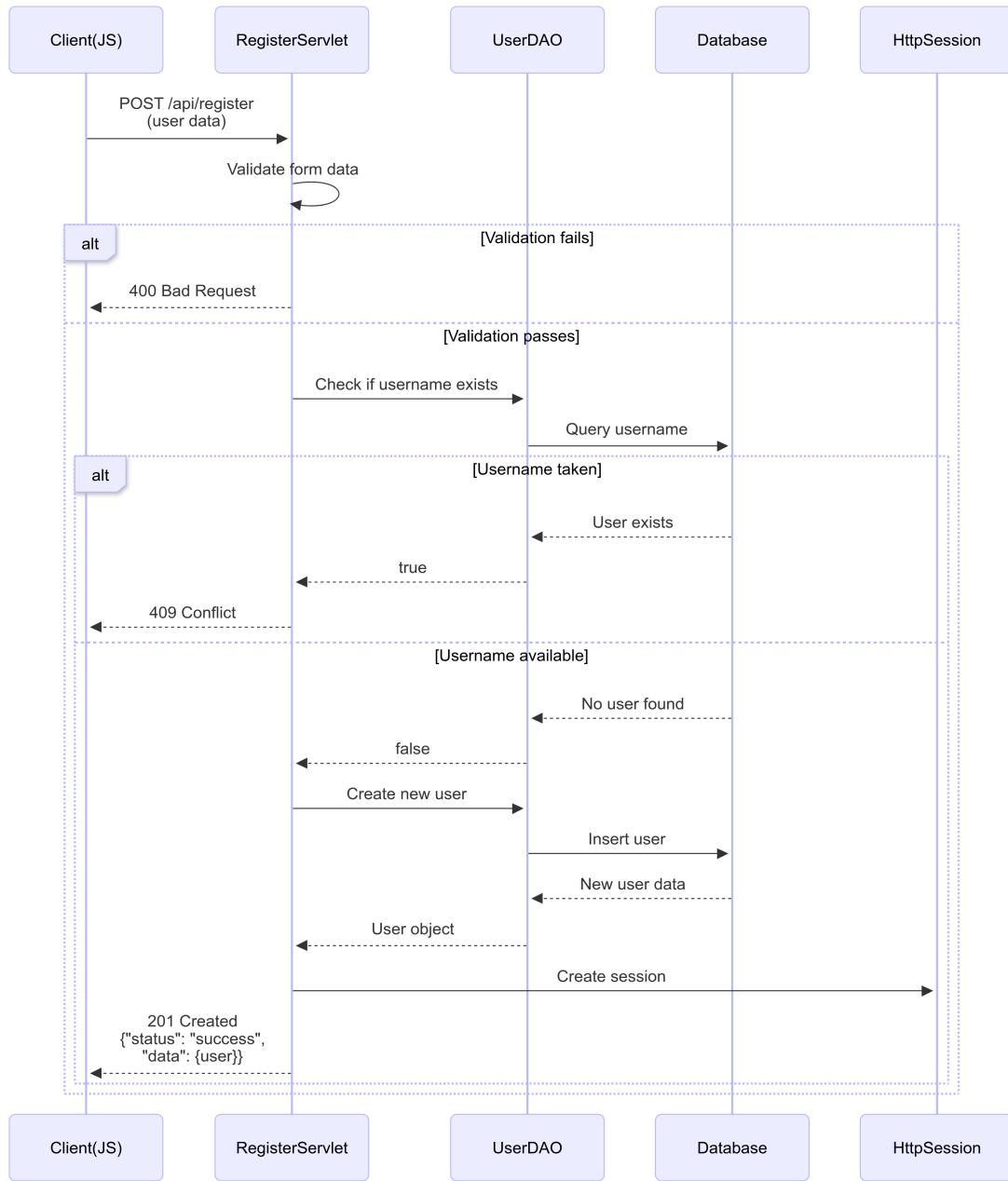
Event: CheckAuthServletRIA - GET /api/checkAuth



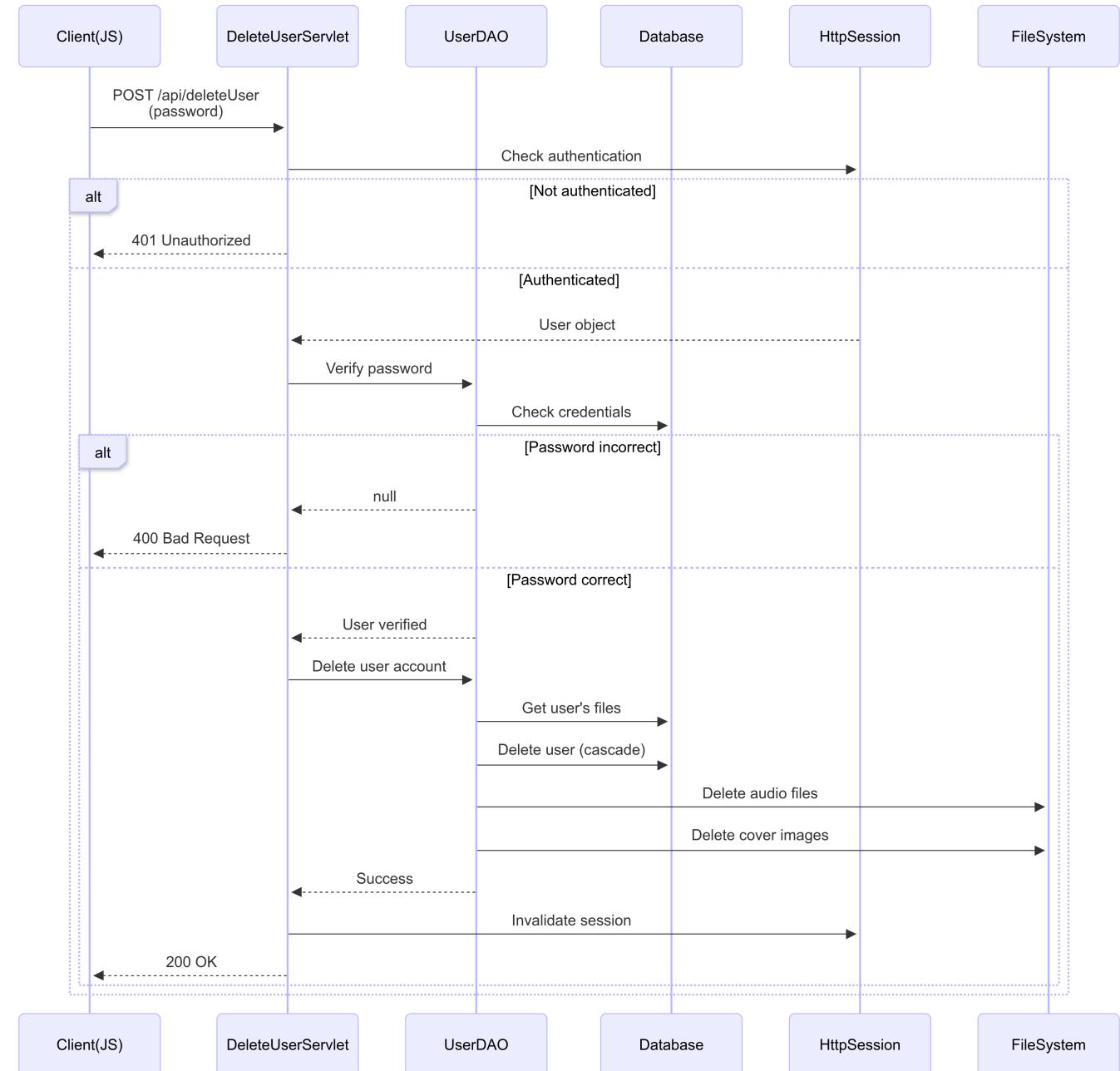
Event: LoginServletRIA - POST /api/login



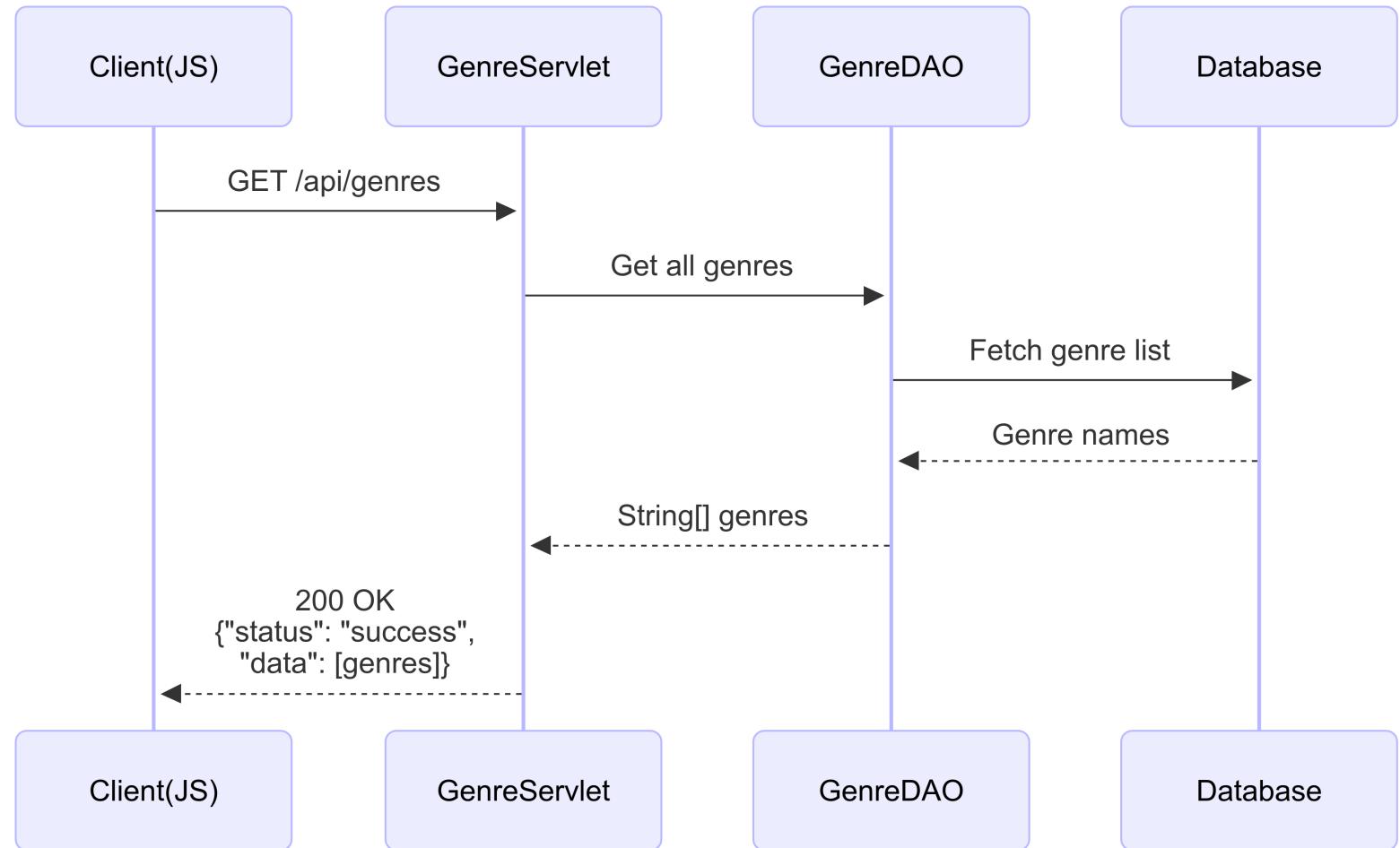
Event: RegisterServletRIA - POST /api/register



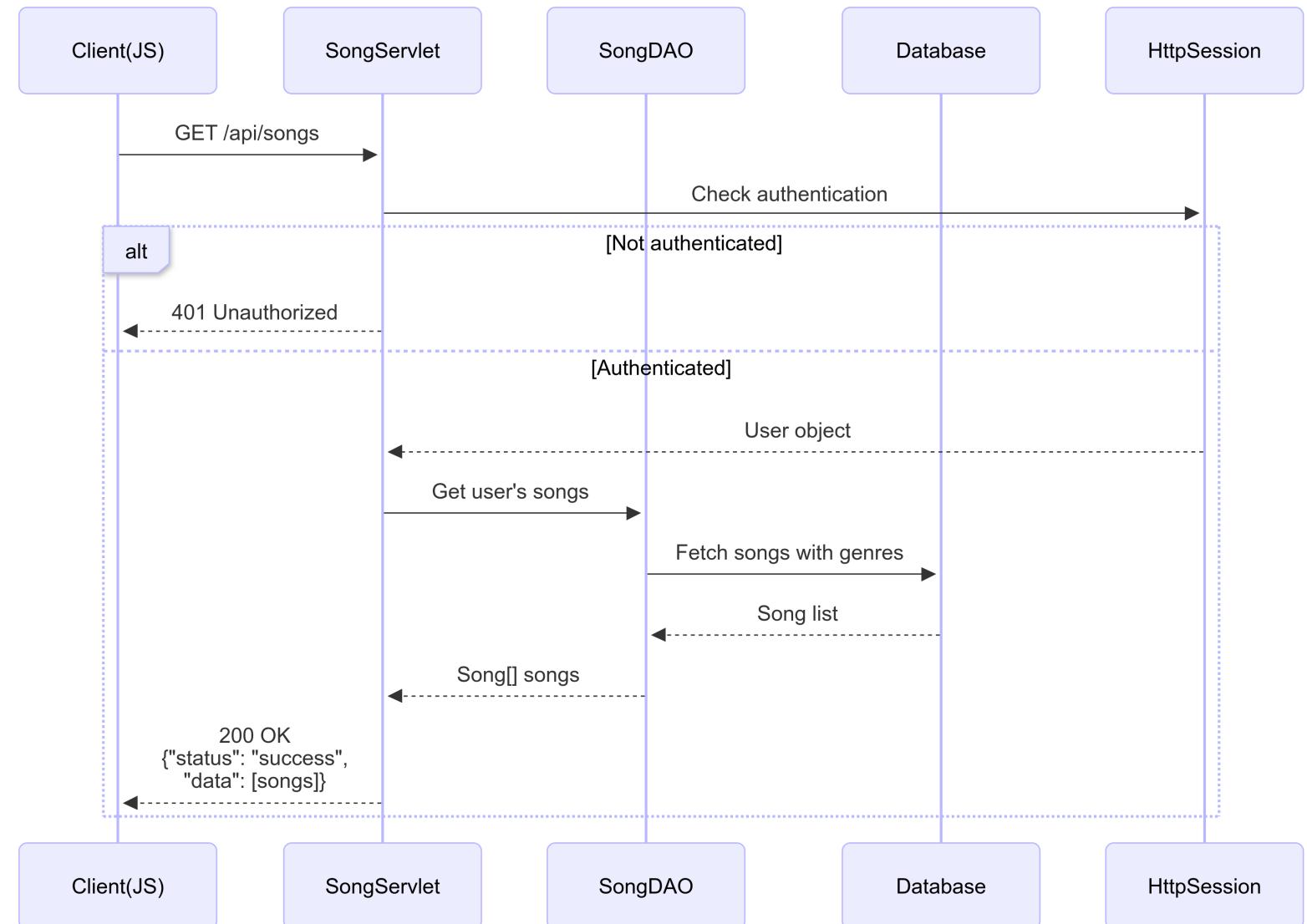
Event: DeleteUserServletRIA - POST /api/deleteUser



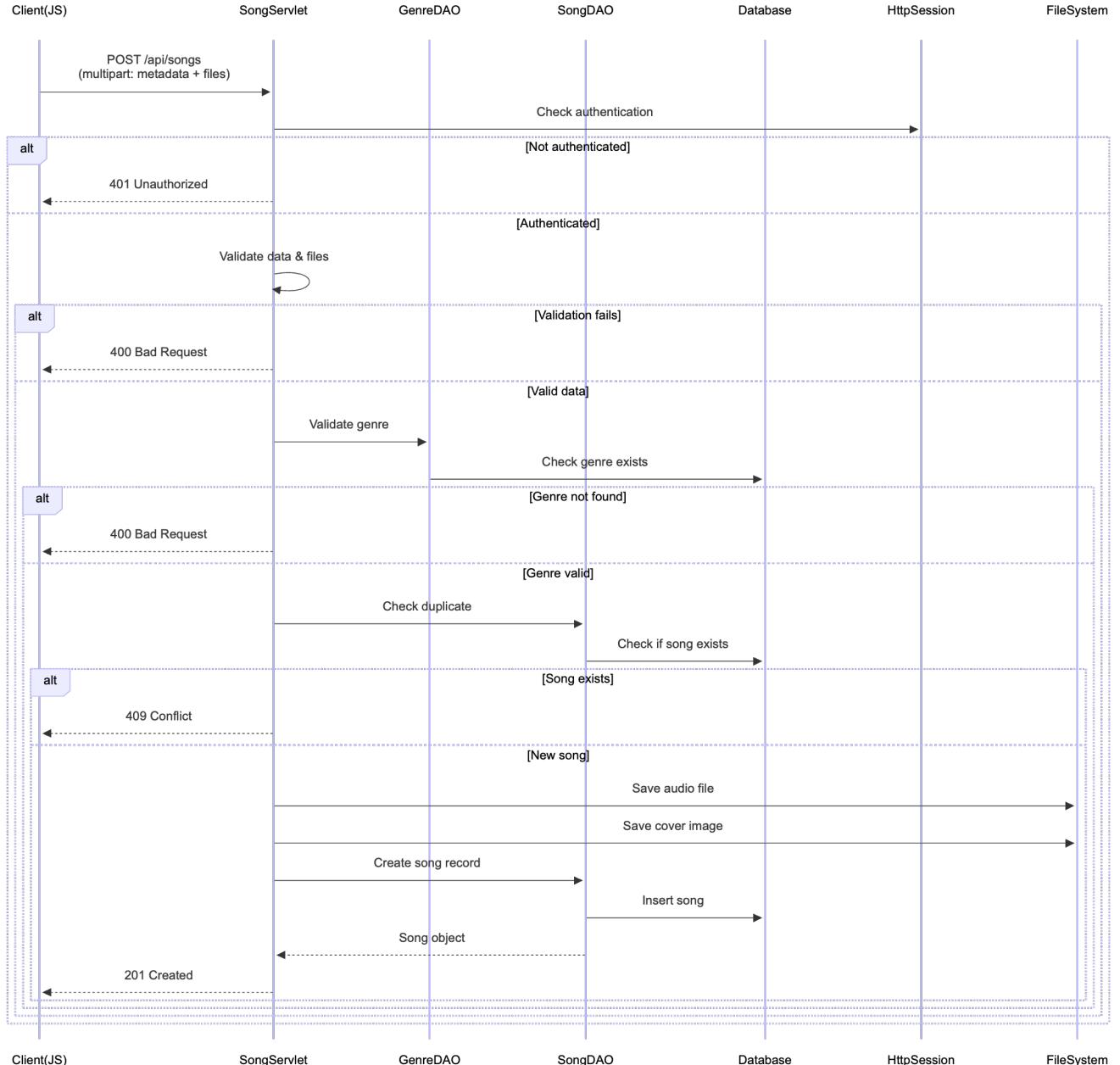
Event: GenreServletRIA - GET /api/genres



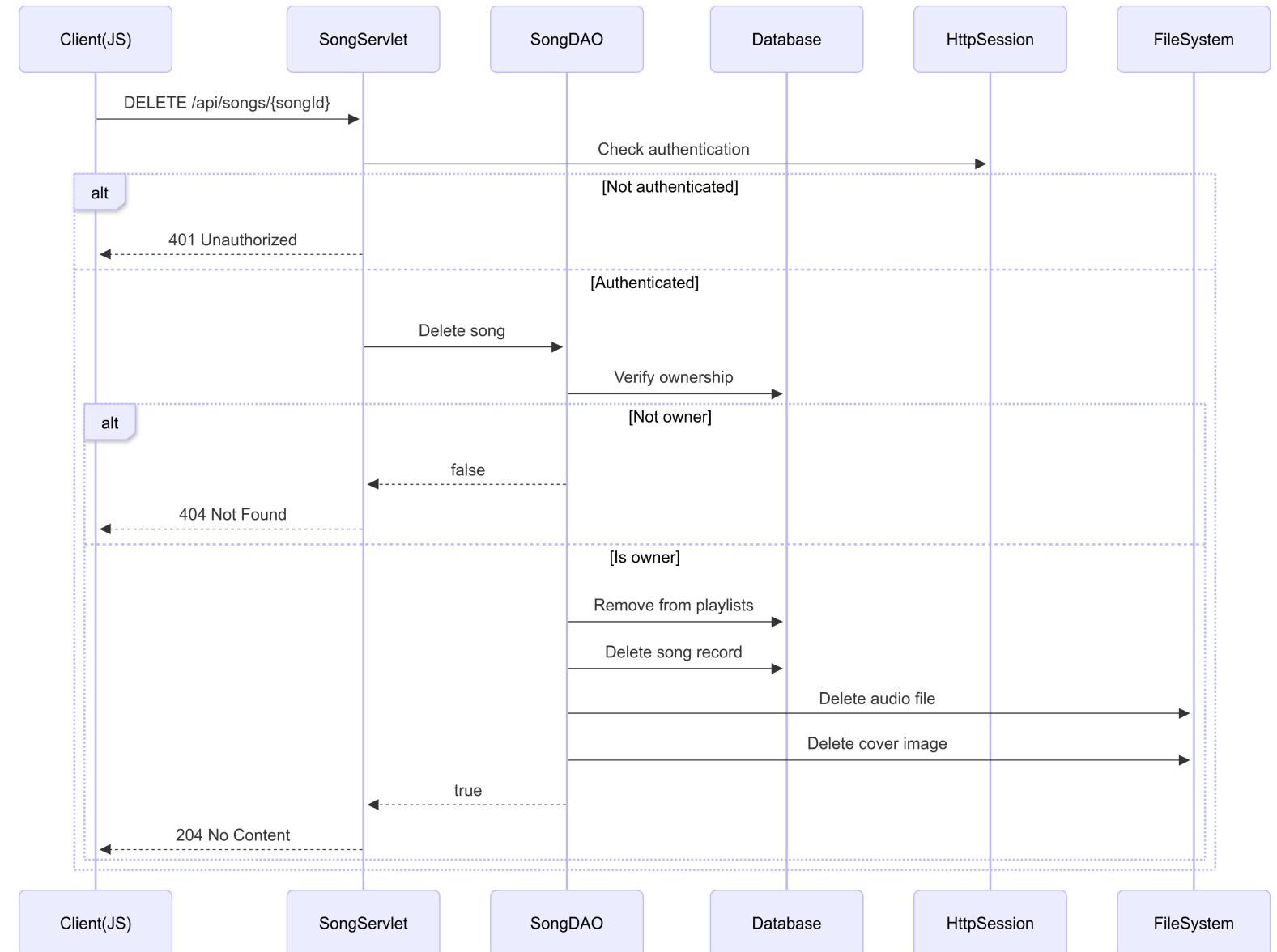
Event: SongServletRIA - GET /api/songs



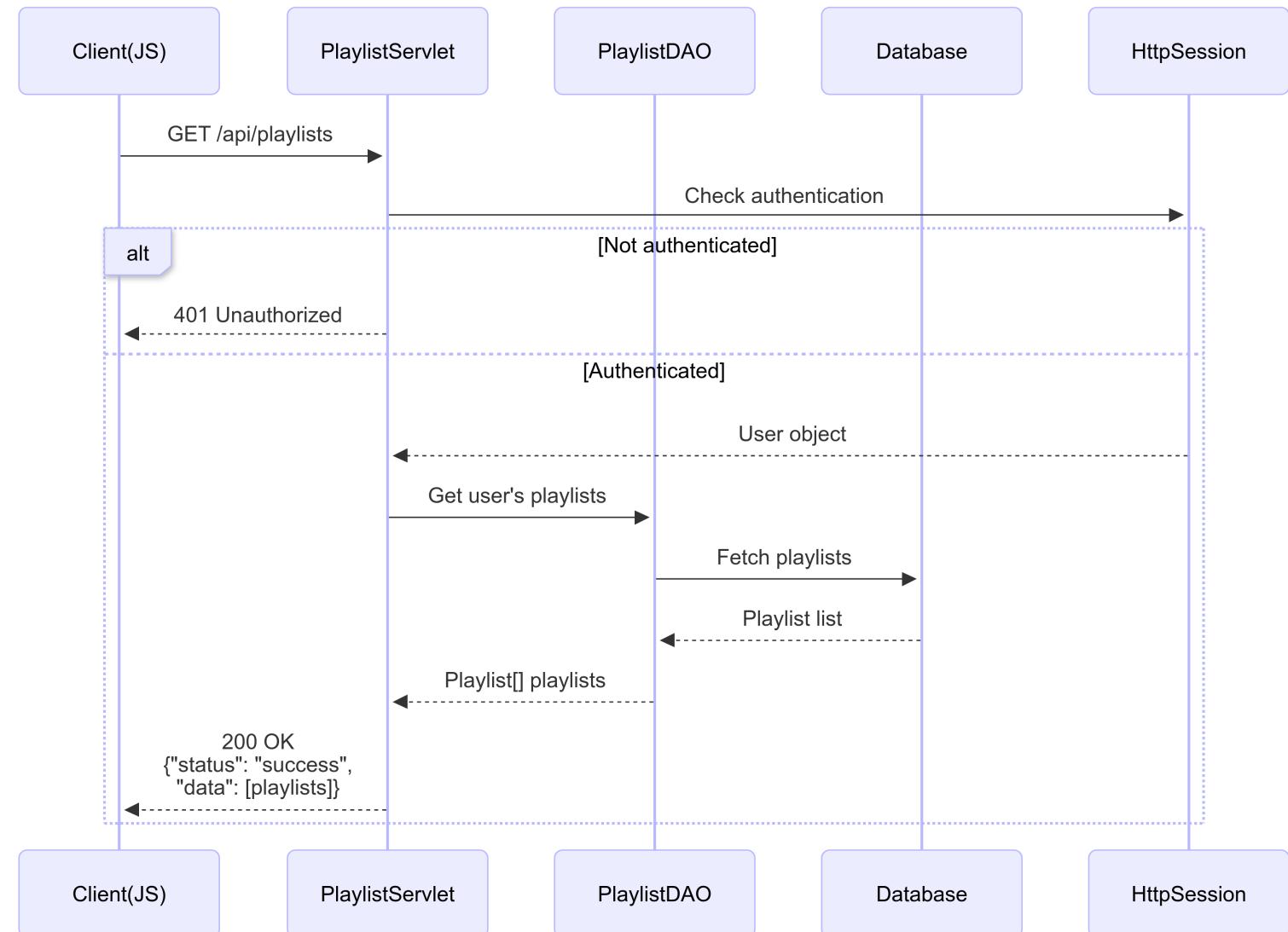
Event: SongServletRIA - POST /api/songs (Upload)



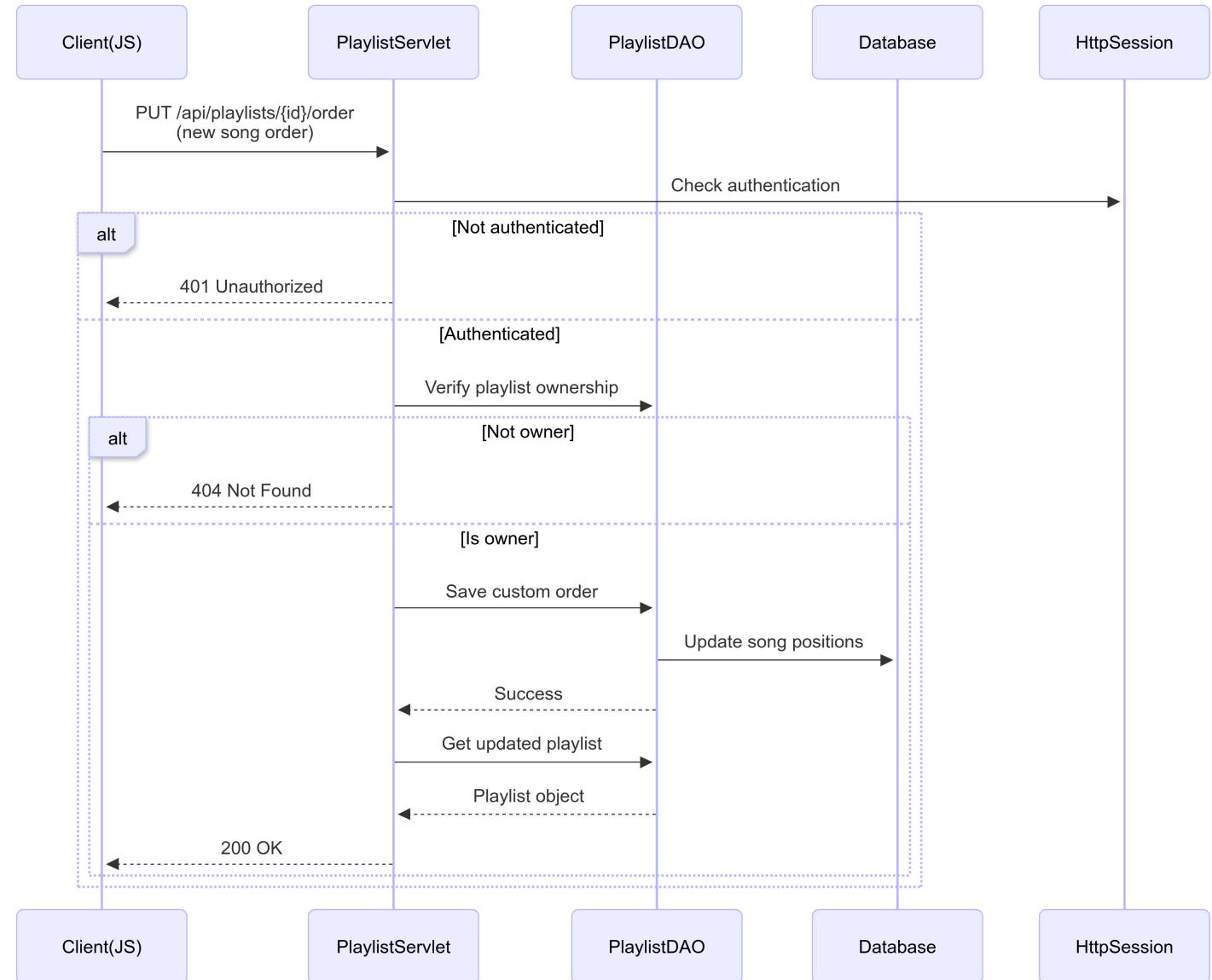
Event: SongServletRIA - DELETE /api/songs/{id}



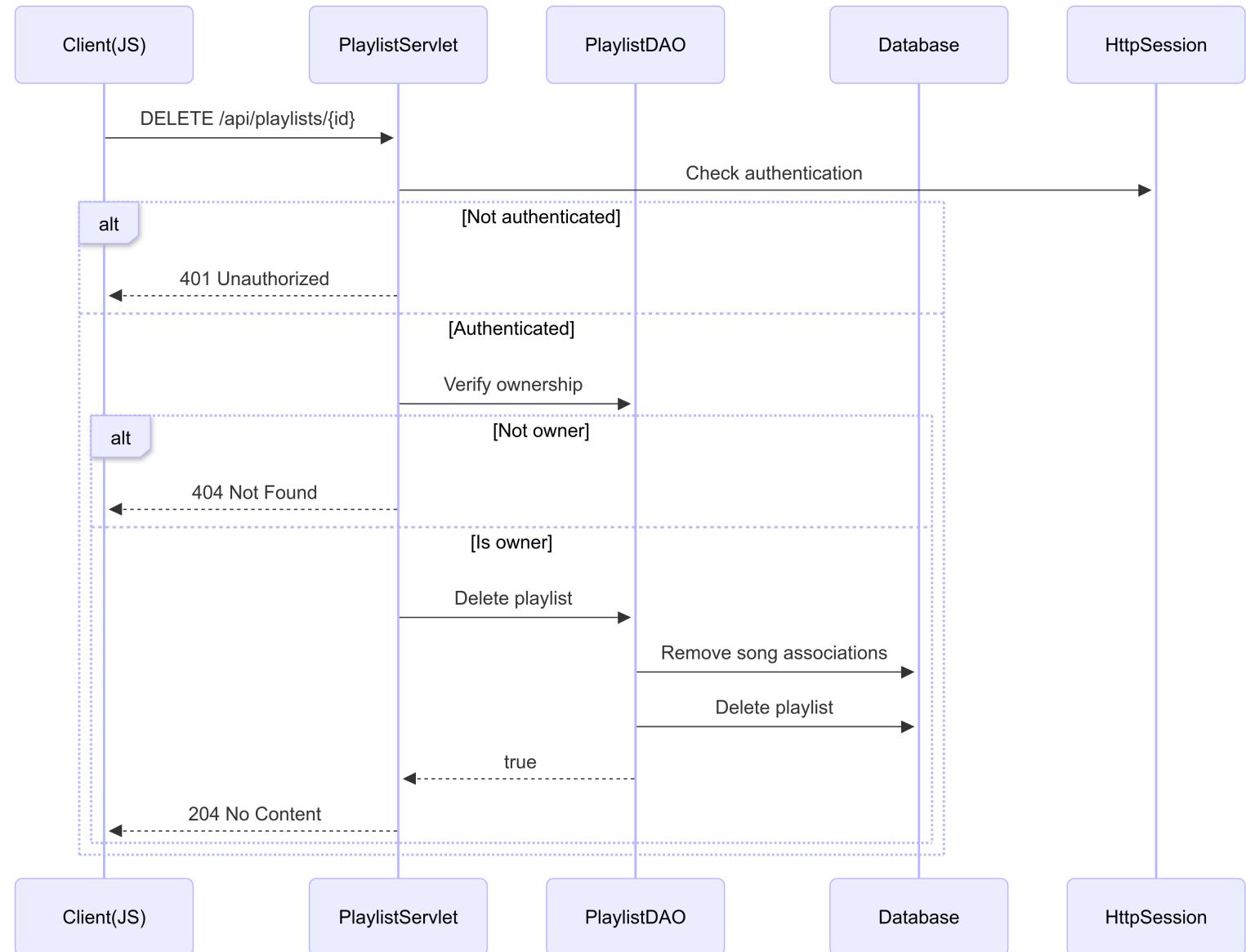
Event: PlaylistServletRIA - GET /api/playlists



Event: PlaylistServletRIA - PUT /api/playlists/{id}/order



Event: PlaylistServletRIA - DELETE /api/playlists/{id}



Event: FileServingServlet - GET /GetFile/*

