Tecnologie informative per il Web @ PoliMi, 2025

Arturo Matteo Santarlasci
arturomatteo.santarlasci@mail.polimi.it
https://github.com/ArturoMatteoSantarlasci

Filippo Polvani
filippo.polvani@mail.polimi.it
https://github.com/FilippoPolvani

Indice

Abstract

Panoramica TIW Playlist Musicale 2025 è un'applicazione web per la gestione di brani musicali e playlist personali, sviluppata come progetto didattico al **Politecnico di Milano** (Tecnologie Internet e Web). Il codice sorgente è reperibile su GitHub. L'app permette agli utenti di registrarsi e autenticarsi, caricare i propri brani musicali (inclusi file audio e copertine) e organizzare questi brani in playlist personalizzate. Le funzionalità includono la creazione di nuove playlist, l'aggiunta/rimozione di brani dalle playlist, la riproduzione audio con un mini-player integrato e la ricerca/filtraggio di brani.

Il progetto è in realtà formato da due sottoprogetti: una versione **pure_HTML**, strutturata come una serie di pagine web separate, e una versione **RIA**, strutturata come una webapp a pagina singola con aggiornamenti dinamici lato client. Le funzionalità sono pressocchè le stesse; le differenze di codice riguardano principalemnte il livello di frontend.

Strumenti Sono state adottate le seguenti tecnologie: **Java** per il backend del server, sfruttando le API di Jakarta; **Apache Tomcat** per l'esecuzione del server; per la versione HTML, **Thymeleaf**, un motore di template; mentre per la versione RIA, **Javascript**. Per il database, **MariaDB** è stato scelto come DBMS, e la comunicazione con il server avviene tramite **JDBC**.

La documentazione è stata realizzata mediante l'utilizzo di **Typst**, il successore di LaTeX. Inoltre, per la creazione dei diagrammi sequenziali è stato utilizzato il pacchetto `chronos`.

È un progetto Maven, il quale scaricherà automaticamente tutte le dipendenze corrette (come Thymeleaf e il driver JDBC). È stato scelto di utilizzare **IntelliJ IDEA Ultimate Edition**. Dopo aver verificato che tutte le dipendenze siano state installate correttamente, esegui la configurazione di Tomcat desiderata e lascia che il server venga distribuito. Sarà accessibile all'indirizzo:

http://localhost:8080/[version]_war_exploded: `[version]` può essere `pure_html` oppure `js`

I file degli **upload** vengono cariati in una cartella interna al computer, in locale, così da non salvarli nel codice sorgente.

Di seguito vengono presentate le specifiche, l'architettura del database, il riepilogo del codice per entrambe le versioni, alcuni diagrammi sequenziali delle principali interazioni, i filtri applicativi, la gestione dello stile (CSS) e una panoramica delle tecnologie utilizzate.

1

Consegna progetto

1.1 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'abum 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'abum 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.

1.2 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 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

Specifiche

2.1 Requisiti Funzionali

L'applicazione consente agli utenti di registrarsi con un nickname univoco e autenticarsi tramite login. Dopo il login, l'utente può caricare nuovi brani musicali sul sistema (inserendo titolo, artista, album, anno, genere e file audio più copertina). I brani caricati sono associati al profilo dell'utente. L'utente può creare playlist personalizzate assegnando un nome alla playlist (univoco per utente) e selezionando uno o più brani da aggiungere. Nella versione in RIA è presente una funzionalità di **«reorder»** che permette di riordinare appunto i brani nella playlist. Cliccando su una playlist, l'utente vede i brani in essa contenuti (5 brani per pagina), con la possibilità di avviare la riproduzione di un brano grazie a un player audio integrato (pagina di player dedicata o "mini-player" sovrapposto).

L'interfaccia offre notifiche di feedback all'utente, ad esempio messaggi di conferma tramite **toast** (piccoli popup) dopo una registrazione avvenuta, creazione di playlist o caricamento di brano riuscito, oppure messaggi di errore in caso di credenziali errate o dati non validi.

2.2 Requisiti non funzionali

L'applicazione è stata progettata ponendo attenzione a usabilità e responsività. L'interfaccia utente adotta un tema grafico ispirato ad **Apple Music**, con design **responsive** (layout adattivo e uso di unità relative/clamp per dimensioni di font e spaziature) e «componenti riutilizzabili» (ad esempio il mini-player flottante presente su più pagine, e modali riutilizzate per inserimento di dati).

Il sistema garantisce buone prestazioni grazie alla versione RIA che riduce i caricamenti completi di pagina utilizzando chiamate AJAX asincrone: questo migliora la reattività percepita, ad esempio quando si aggiungono brani a una playlist o si carica un nuovo brano, gli aggiornamenti avvengono senza ricaricare l'intera pagina.

Dal punto di vista della sicurezza e consistenza dei dati, sono implementati controlli sia lato client sia lato server: i form HTML utilizzano attributi di validazione (es. required, minlength, formato type=email/number etc.), mentre lato server sono presenti controlli sui parametri (ad es. verifica che l'anno sia un numero valido nell'intervallo atteso) e vincoli a livello di database (es. vincoli di unicità e integrità referenziale).

Inoltre, appositi **filtri** prevengono accessi non autorizzati e garantiscono che ogni utente interagisca solo con le proprie risorse.

Il codice è strutturato per essere **manutenibile**, con separazione in layer (DAO per l'accesso al database, Servlet controller per la logica di business, pagine/JS per la presentazione) e con due implementazioni (HTML e RIA).

3

Database SQL

3.1 Struttura database SQL

I requisiti del progetto variano leggermente tra le versioni`pure_html` e `ria`: in quest'ultima, infatti, è richiesto che i brani possano avere un ordine personalizzato all'interno della playlist a cui appartengono — funzionalità ottenuta tramite una semplice modifica nello schema delle tabelle SQL.

L'applicazione si appoggia dunque a un database relazionale MySQL (schema tiw2025) contenente quattro tabelle principali: **user, track, playlist, playlist_tracks**.

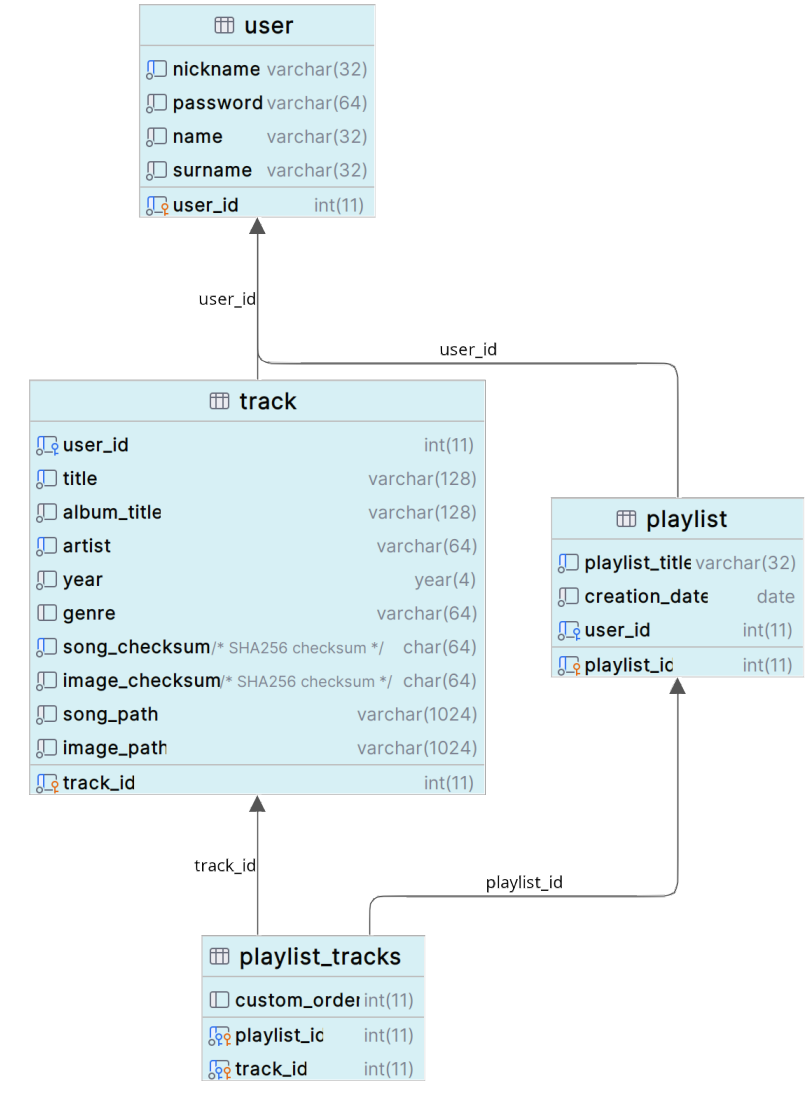


Figura 1: UML diagram

Qui di seguito mostriamo i diagrammi ER per entrambi i progetti; possiamo notare ancora come l'unica variazione tra i due progetti è la presenza di `custom_order` per la versione `RIA`

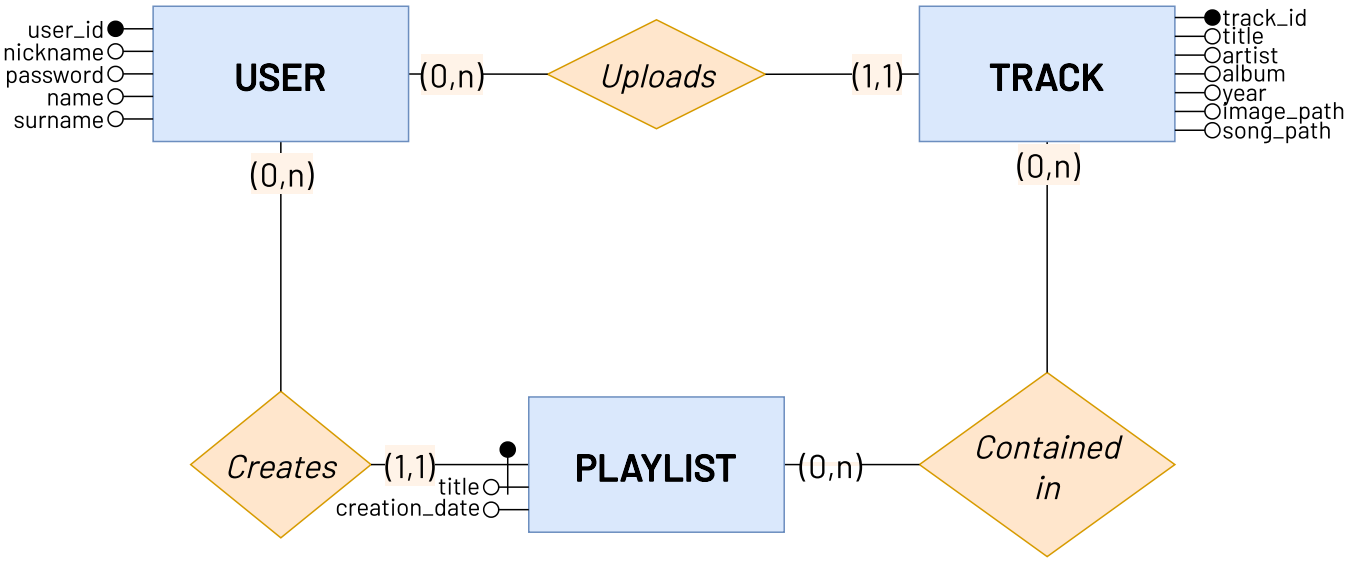


Figura 2: diagramma ER (HTML)

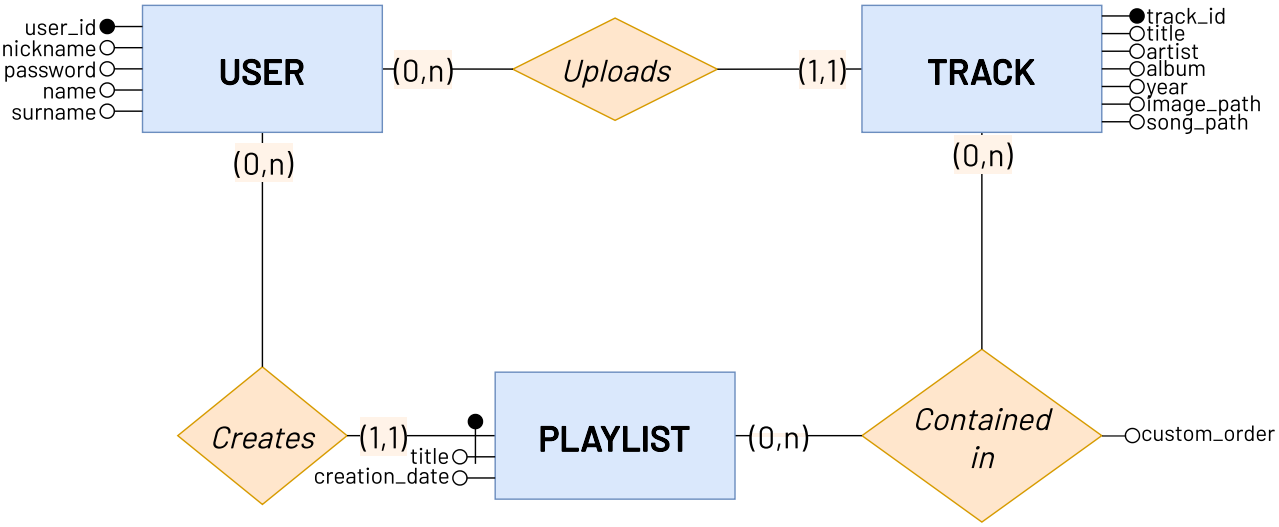


Figura 3: diagramma ER (RIA)

3.2 Tabelle SQL

Mostriamo ora come sono state strutturate le tabelle:

- **user**

```
CREATE TABLE user
(
    user_id  integer     not null auto_increment,
    nickname varchar(32) not null unique,
    password varchar(64) not null,
    name     varchar(32) not null,
    surname  varchar(32) not null,

    primary key (user_id)
);
```

l'attributo `user_id` rappresenta la chiave primaria; l'unico altro attributo con un vincolo di unicità è nickname.

- **track**

```
CREATE TABLE track
(
    track_id       integer       not null auto_increment,
    user_id        integer       not null,
    title          varchar(128)  not null,
    album_title    varchar(128)  not null,
    artist         varchar(64)   not null,
    year           year          not null,
    genre          varchar(64),
    song_checksum  char(64)      not null default '0...0',
    image_checksum char(64)      not null default '0...0',
    song_path      varchar(1024) not null,
    image_path     varchar(1024) not null,

    primary key (track_id),
    foreign key (user_id) REFERENCES user (user_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
    unique (user_id, song_checksum),
    unique (user_id, title, artist),
    check (genre in ('Classical', 'Rock', 'Edm', 'Pop', 'Hip-hop', 'R&B', 'Country', 'Jazz',
'Blues', 'Metal', 'Folk', 'Soul', 'Funk', 'Electronic', 'Indie', 'Reggae', 'Disco'))
);
```

è stato introdotta una funzionalità speciale: il checksum SHA256 per il brano e l'immagine dell'album, che consente al server di salvare una sola copia di ciascun file, evitando duplicati basati solo sul nome.

La tabella include un vincolo di unicità su `user_id`, `song_checksum` per garantire questa proprietà, e su `user_id`, `title`, `artist` per evitare duplicati interni. Ogni traccia è infine collegata in modo univoco al suo utente tramite chiave esterna.[1]

[1] Un utente non può avere tracce duplicate

- **playlist**

```
CREATE TABLE playlist
(
    playlist_id    integer     not null auto_increment,
    playlist_title varchar(32) not null,
    creation_date  date        not null default CURRENT_DATE,
    user_id        integer     not null,

    primary key (playlist_id),
    unique (playlist_title, user_id),
    foreign key (user_id) REFERENCES user (user_id)
        ON DELETE CASCADE ON UPDATE CASCADE
);
```

L'attributo `creation_date` assume come valore predefinito la data odierna; inoltre, il vincolo di unicità su `playlist_title`, `user_id` assicura che ogni utente possa avere una sola playlist con lo stesso titolo, collegata a lui tramite chiave esterna.

- **playlist_tracks** - per pure_HTML

```
CREATE TABLE playlist_tracks - per
(
    playlist_id  integer not null,
    track_id     integer not null,

    primary key (playlist_id, track_id),
    foreign key (playlist_id) REFERENCES playlist (playlist_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
    foreign key (track_id) REFERENCES track (track_id)
        ON DELETE CASCADE ON UPDATE CASCADE
);
```

Questa tabella rappresenta la relazione "Contained in" nel diagramma ER (Figura 2). La sua chiave primaria è composta (l'unica del progetto) e serve a collegare un brano a una playlist. A differenza delle altre tabelle, che richiedevano una chiave primaria e un vincolo di unicità distinti, qui una chiave composta è la scelta corretta, poiché uno stesso brano può appartenere a più playlist.

- JS **playlist_tracks** - per RIA

```
CREATE TABLE playlist_tracks
(
    playlist_track_id integer auto_increment,
    playlist_id       integer not null,
    track_id          integer not null,
    custom_order      integer,

    primary key (playlist_track_id),
    unique (playlist_id, track_id),
    foreign key (playlist_id) references playlist (playlist_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
    foreign key (track_id) references track (track_id)
        ON DELETE CASCADE ON UPDATE CASCADE
);
```

Analogamente al codice precedente, anche questa tabella rappresenta la relazione “Contained in” nel diagramma ER della versione RIA (Figura 3), con l’aggiunta dell’attributo custom_order. La precedente chiave primaria è stata convertita in vincolo di unicità, come in altre parti del progetto, mentre il resto della struttura rimane invariato.

4

Struttura del codice

4.1 Componenti

Backend

1. DAOs
   - DAO interface
   - PlaylistDAO
   - TrackDAO
   - UserDAO
2. Entities
   - Playlist
   - Track
   - User
3. Servlets/Controller
   - LoginController
   - HomePageController
   - AddTracksToPlaylist
   - PlaylistController
   - RegisterController
   - MediaServlet
   - PlayerController
   - Logout
   - CreatePlaylist
   - UploadTrack
   - JS GetTracksNotInPlaylist[2]
   - JS GetUserTracks
   - JS TrackReorder
   - JS PlaylistGroupController
4. Filters
   - InvalidUserChecker
   - PlaylistChecker
   - SelectedTracksChecker
   - TrackChecker
   - UserChecker
5. Utils
   - ConnectionHandler
   - TemplateThymeleaf[3]
6. Config
   - MediaConfig

Frontend

Nel frontend, a differenza del backend in cui troviamo solo poche differenze, pure_html e RIA sono molto diverse.

- Pure_HTML
  - HomePage.html
  - Login.html
  - player_page.html
  - playlist_page.html
  - register.html
- JS RIA version
  - HomePage.html
    - HomePage.js
  - login.html
    - login.js
  - register.html
    - register.js
  - utils.js

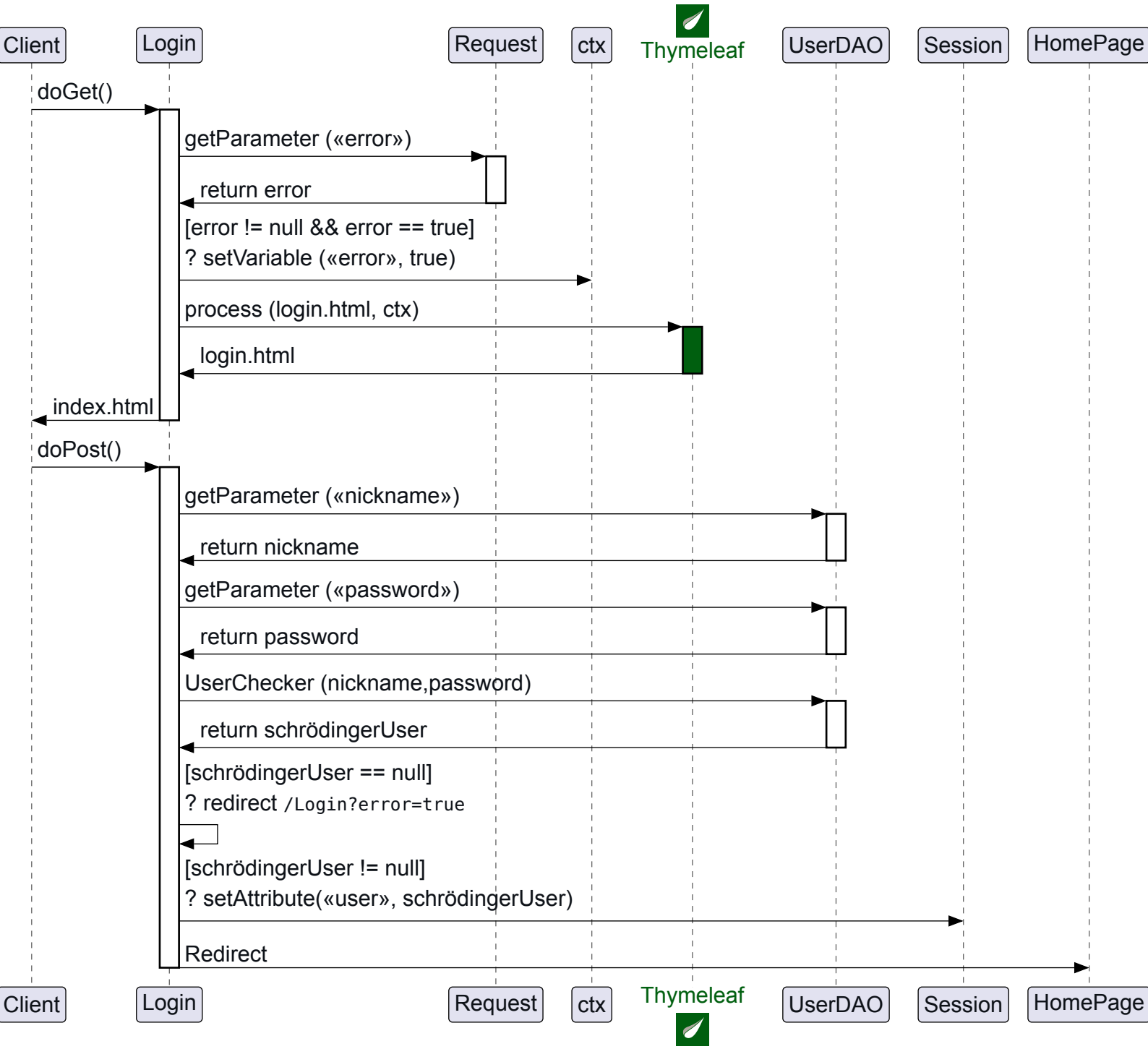
[2] JS Presenti solo nella versione RIA

[3] Solo per pure_html

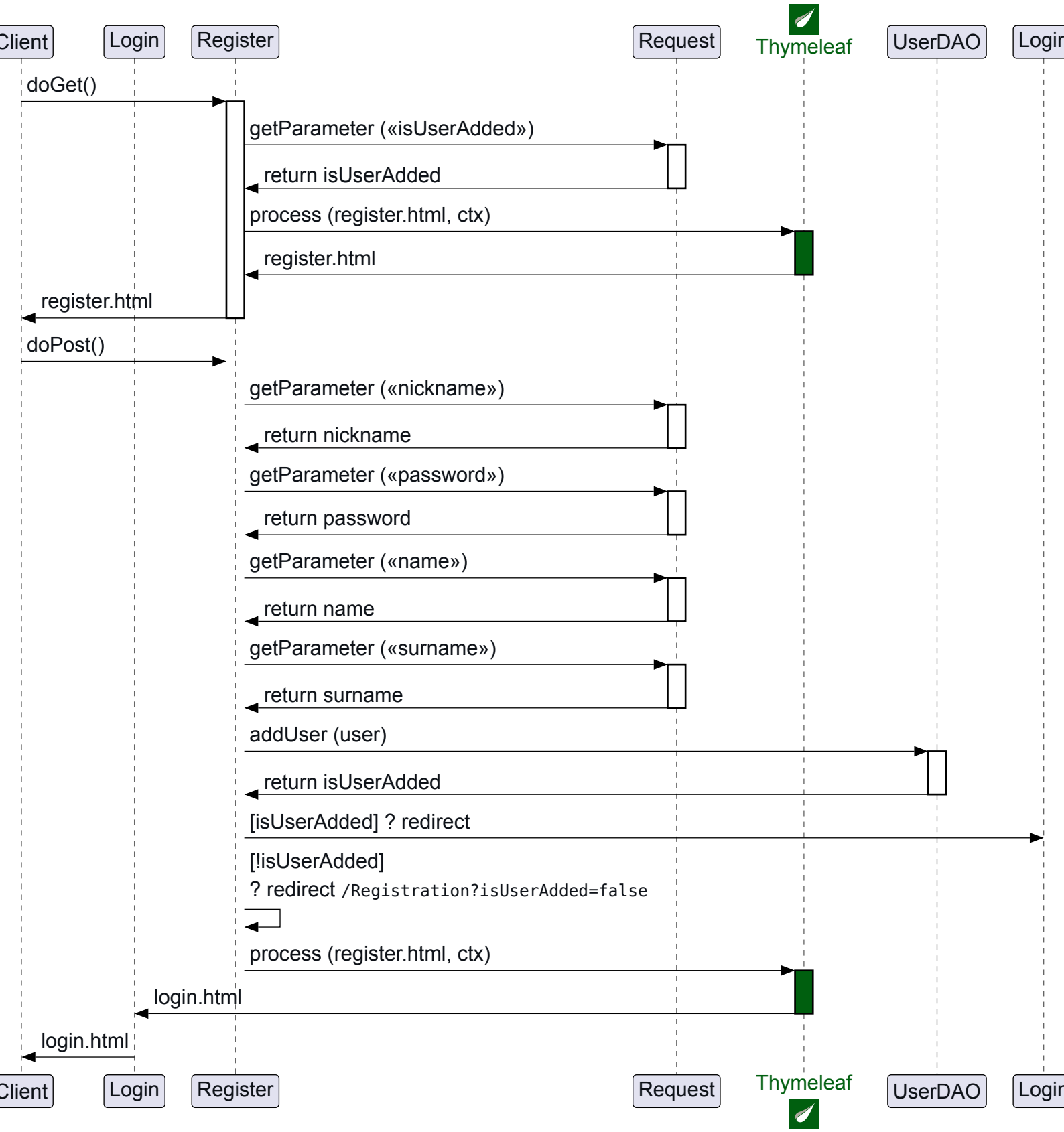
5

Diagrammi di sequenza

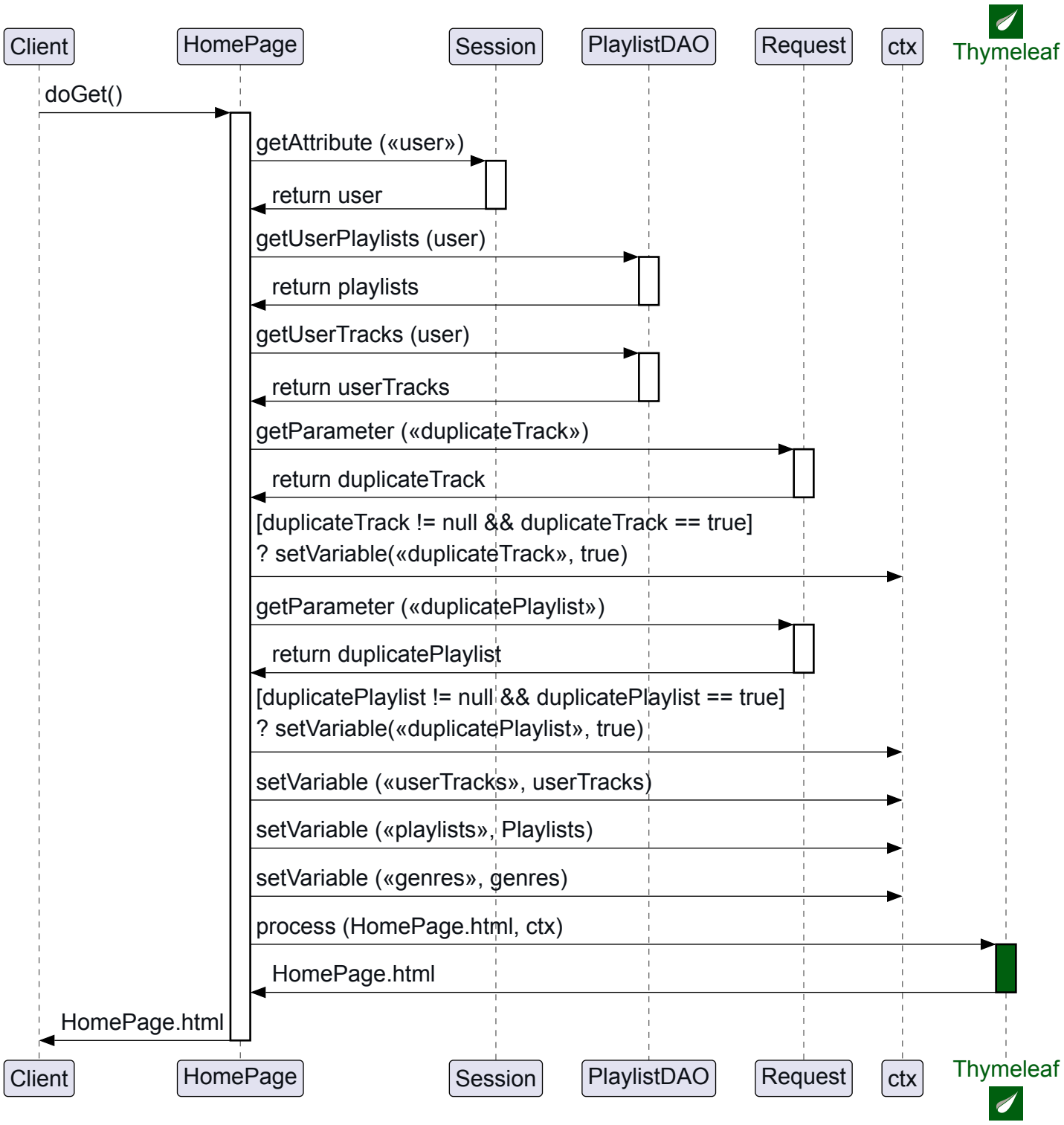
5.1 Login


Client
Login
Request
ctx
Thymeleaf
UserDAO
Session
HomePage
doGet()
getParameter («error»)
return error
[error != null && error == true]
? setVariable («error», true)
process (login.html, ctx)
login.html
index.html
doPost()
getParameter («nickname»)
return nickname
getParameter («password»)
return password
UserChecker (nickname,password)
return schrödingerUser
[schrödingerUser == null]
? redirect /Login?error=true
[schrödingerUser != null]
? setAttribute(«user», schrödingerUser)
Redirect

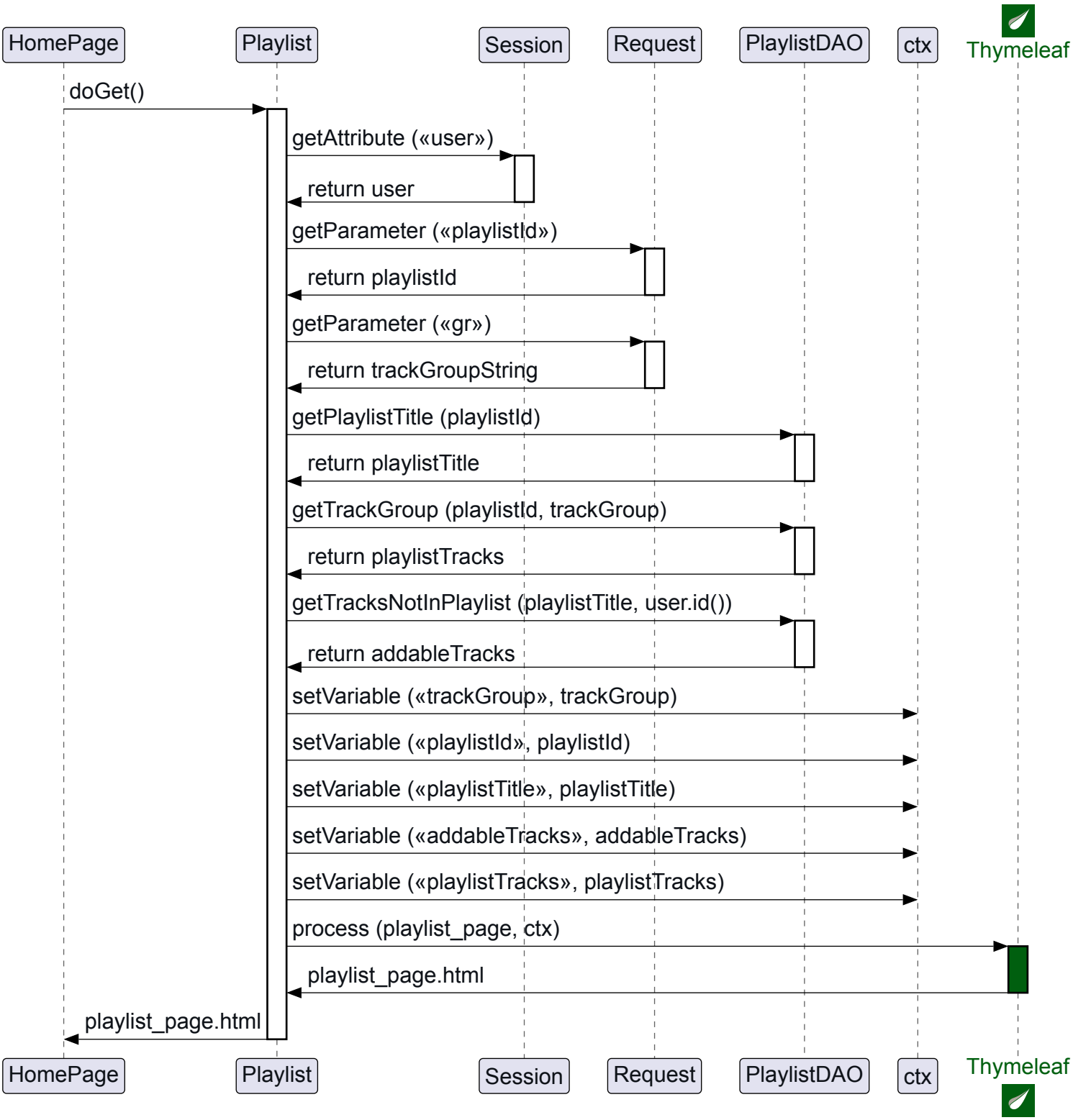
5.2 Registrazione



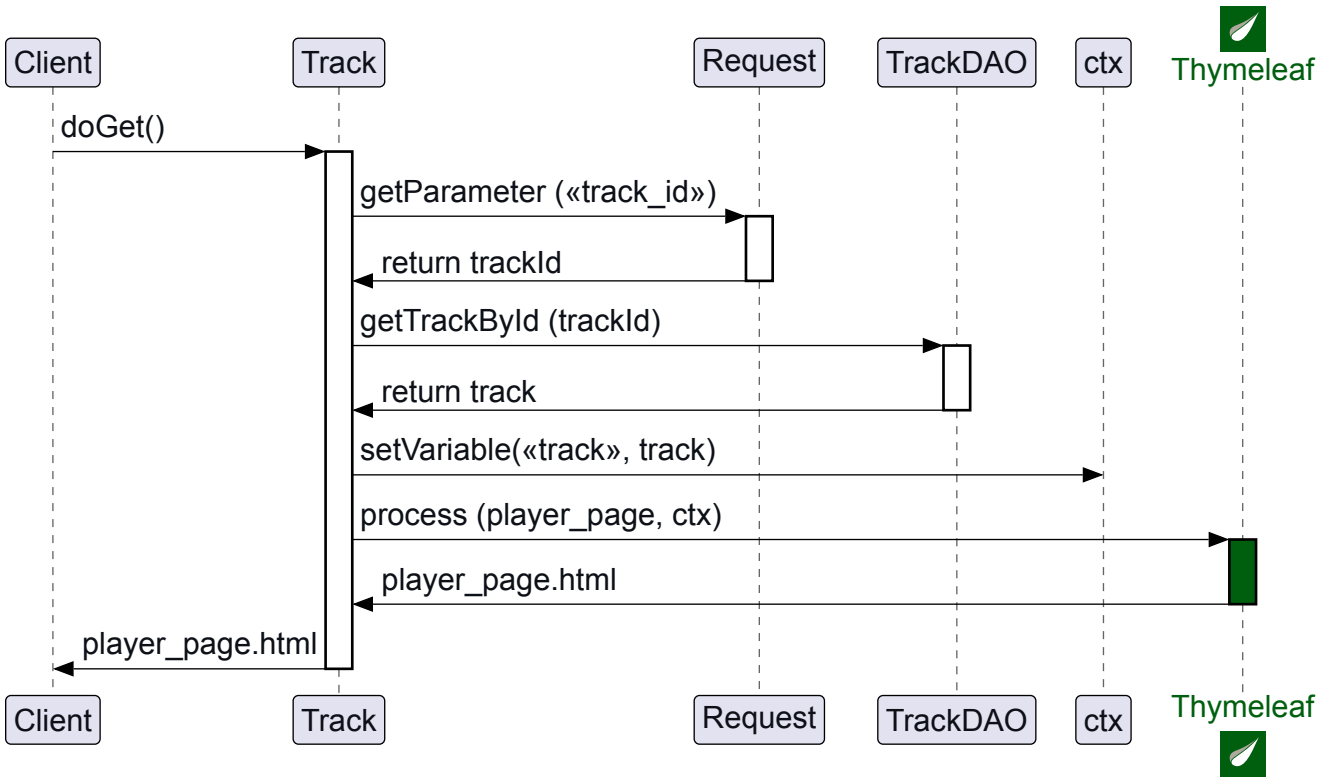
5.3 HomePage


Client
HomePage
Session
PlaylistDAO
Request
ctx
Thymeleaf
doGet()
getAttribute («user»)
return user
getUserPlaylists (user)
return playlists
getUserTracks (user)
return userTracks
getParameter («duplicateTrack»)
return duplicateTrack
[duplicateTrack != null && duplicateTrack == true]
? setVariable(«duplicateTrack», true)
getParameter («duplicatePlaylist»)
return duplicatePlaylist
[duplicatePlaylist != null && duplicatePlaylist == true]
? setVariable(«duplicatePlaylist», true)
setVariable («userTracks», userTracks)
setVariable («playlists», Playlists)
setVariable («genres», genres)
process (HomePage.html, ctx)
HomePage.html
HomePage.html
Client
HomePage
Session
PlaylistDAO
Request
ctx
Thymeleaf

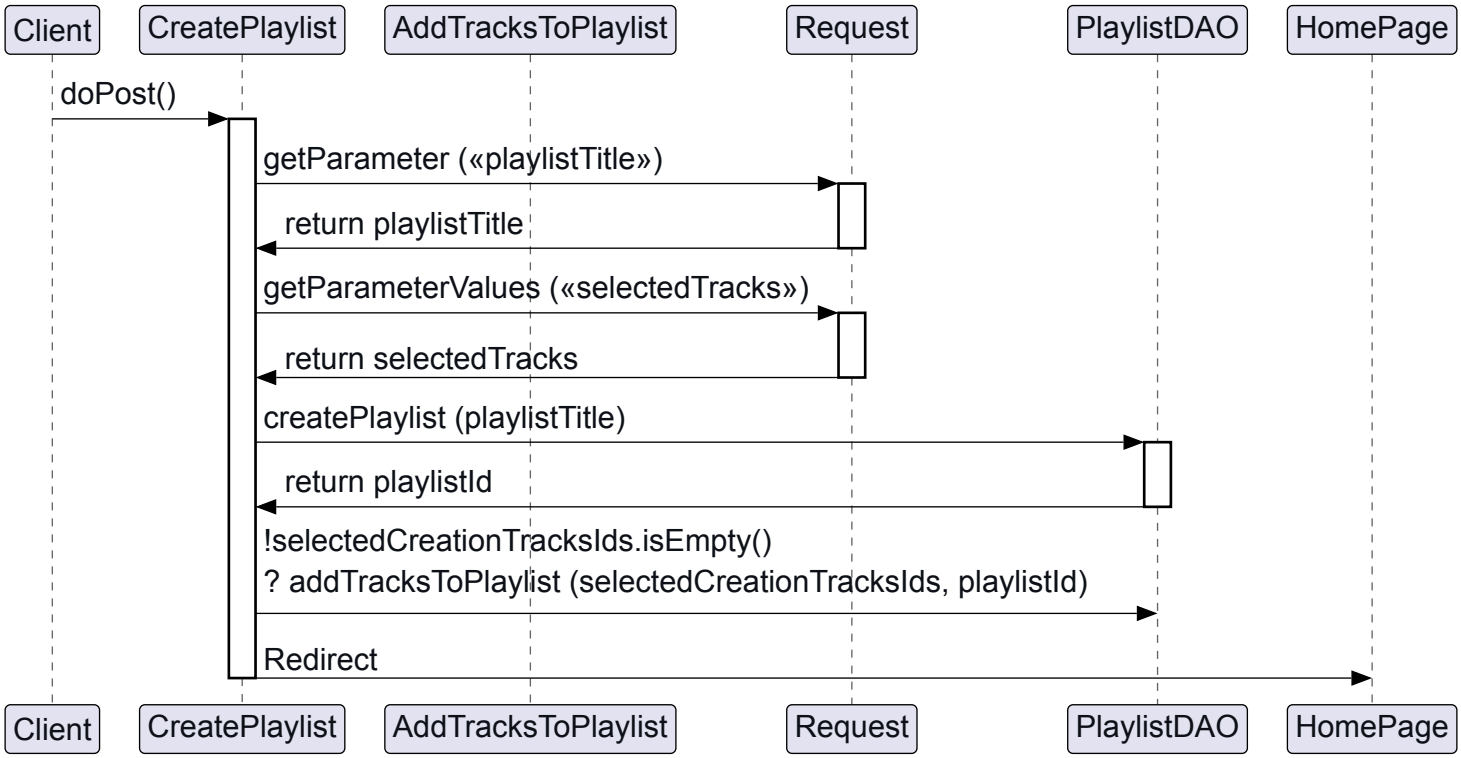
5.4 Playlist


HomePage
Playlist
Session
Request
PlaylistDAO
ctx
Thymeleaf
doGet()
getAttribute («user»)
return user
getParameter («playlistId»)
return playlistId
getParameter («gr»)
return trackGroupString
getPlaylistTitle (playlistId)
return playlistTitle
getTrackGroup (playlistId, trackGroup)
return playlistTracks
getTracksNotInPlaylist (playlistTitle, user.id())
return addableTracks
setVariable («trackGroup», trackGroup)
setVariable («playlistId», playlistId)
setVariable («playlistTitle», playlistTitle)
setVariable («addableTracks», addableTracks)
setVariable («playlistTracks», playlistTracks)
process (playlist_page, ctx)
playlist_page.html
playlist_page.html
HomePage
Playlist
Session
Request
PlaylistDAO
ctx
Thymeleaf

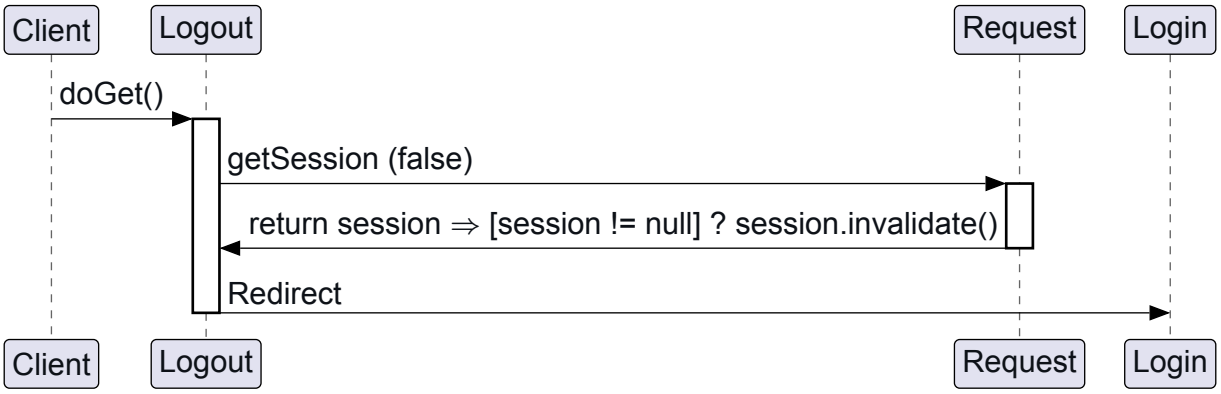
5.5 Track


Client
Track
Request
TrackDAO
ctx
Thymeleaf
doGet()
getParameter («track_id»)
return trackId
getTrackById (trackId)
return track
setVariable(«track», track)
process (player_page, ctx)
player_page.html
player_page.html
Client
Track
Request
TrackDAO
ctx
Thymeleaf

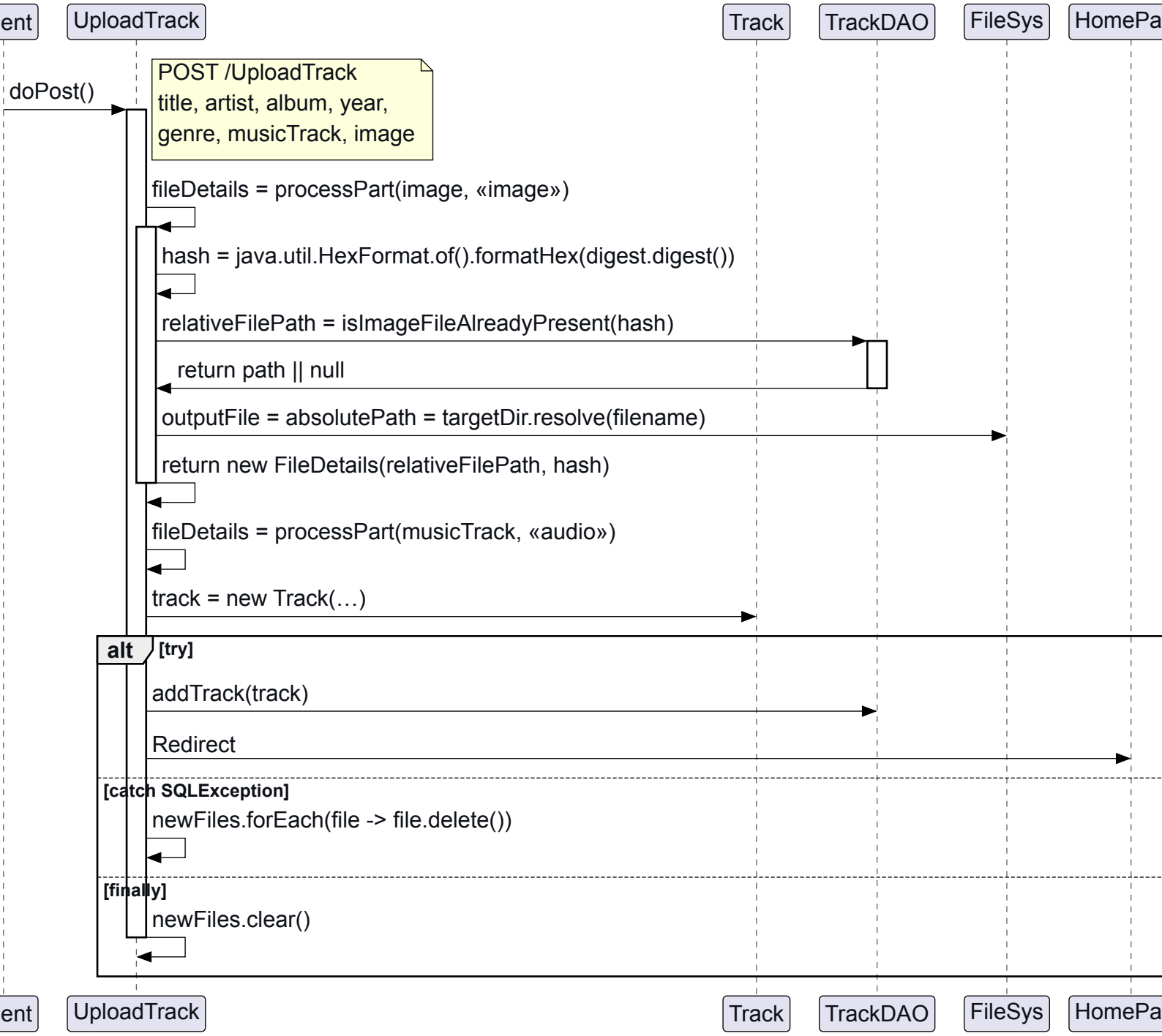
5.6 Creazione Playlist



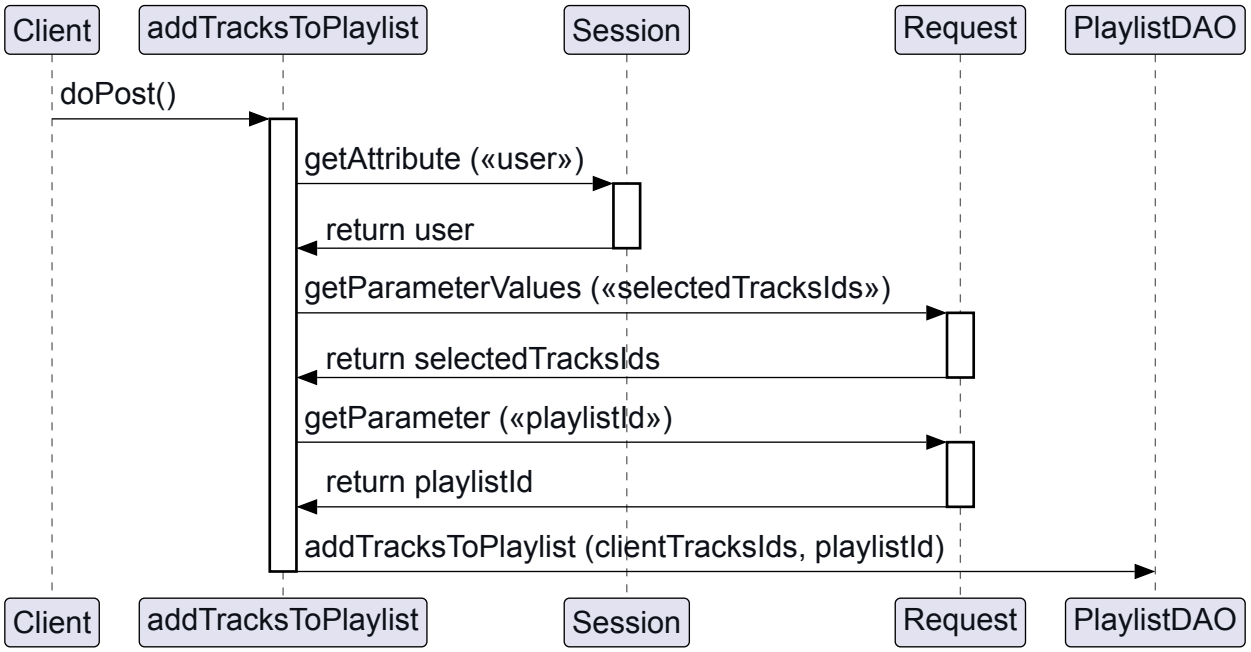
5.7 Logout


Client
Logout
Request
Login
doGet()
getSession (false)
return session ⇒ [session != null] ? session.invalidate()
Redirect
Client
Logout
Request
Login

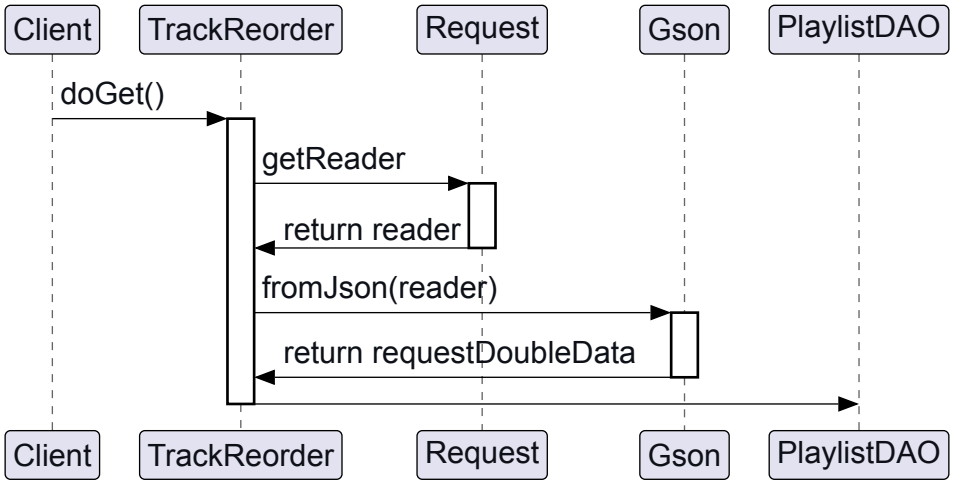
5.8 UploadTrack sequence diagram



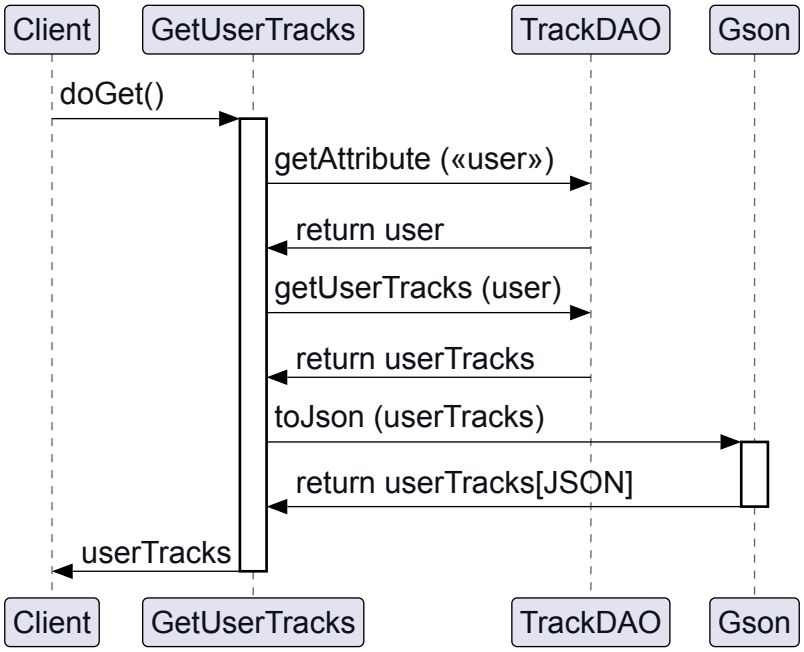
5.9 Aggiungi traccia



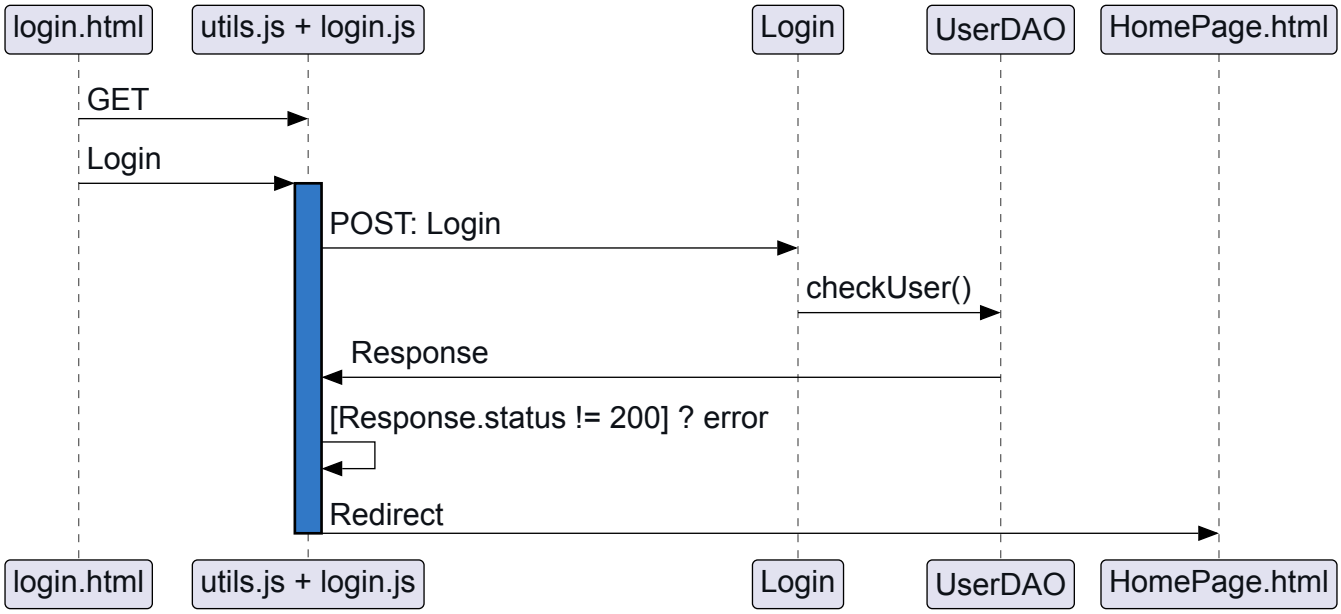
5.10 JS Riordine tracce



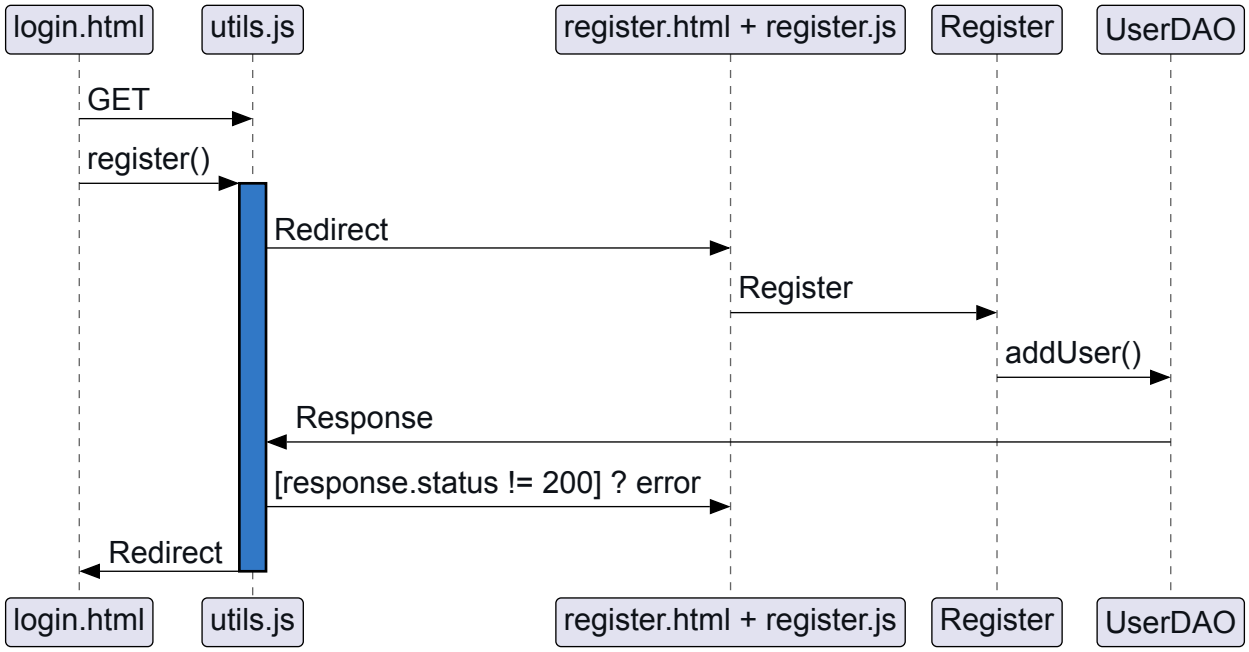
5.11 JS GetUserTracks



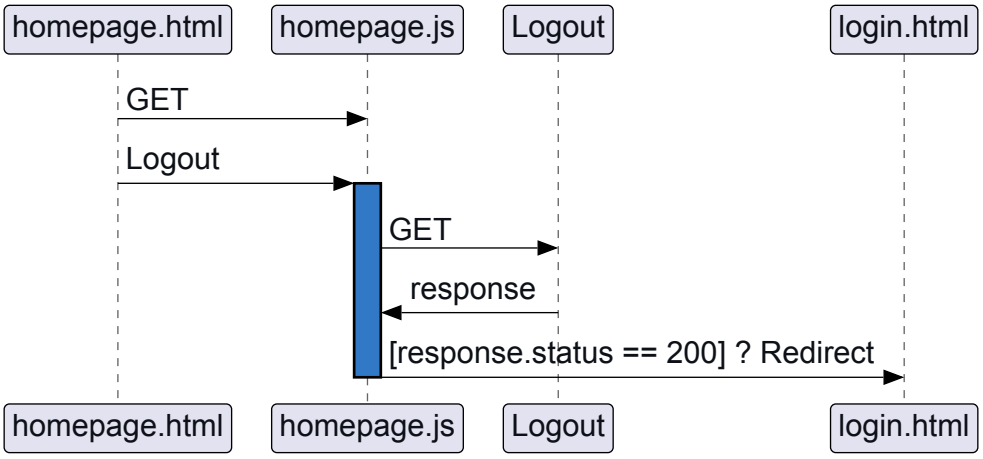
5.12 JS Evento: Login



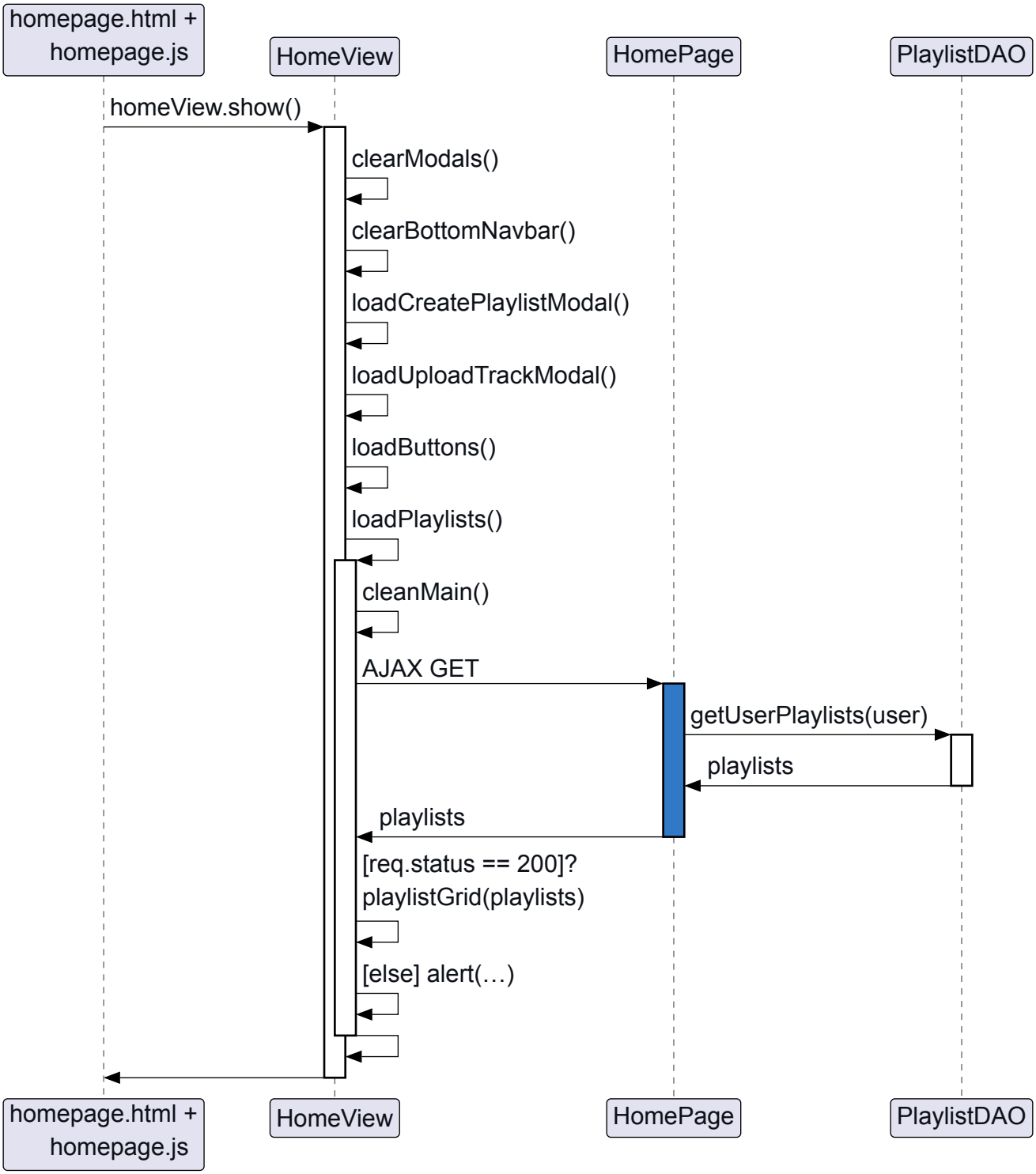
5.13 JS Evento: Register



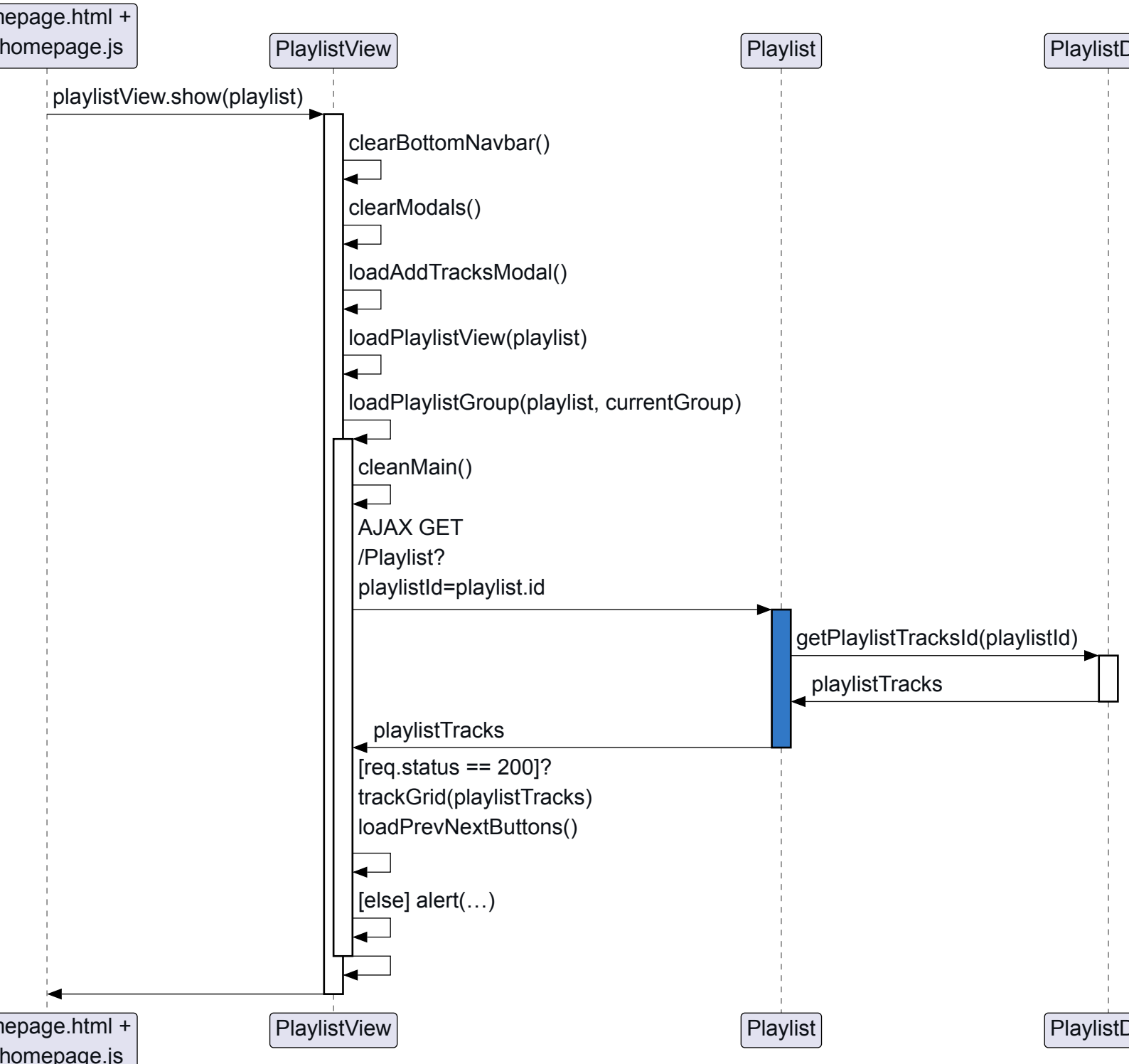
5.14 JS Evento: Logout



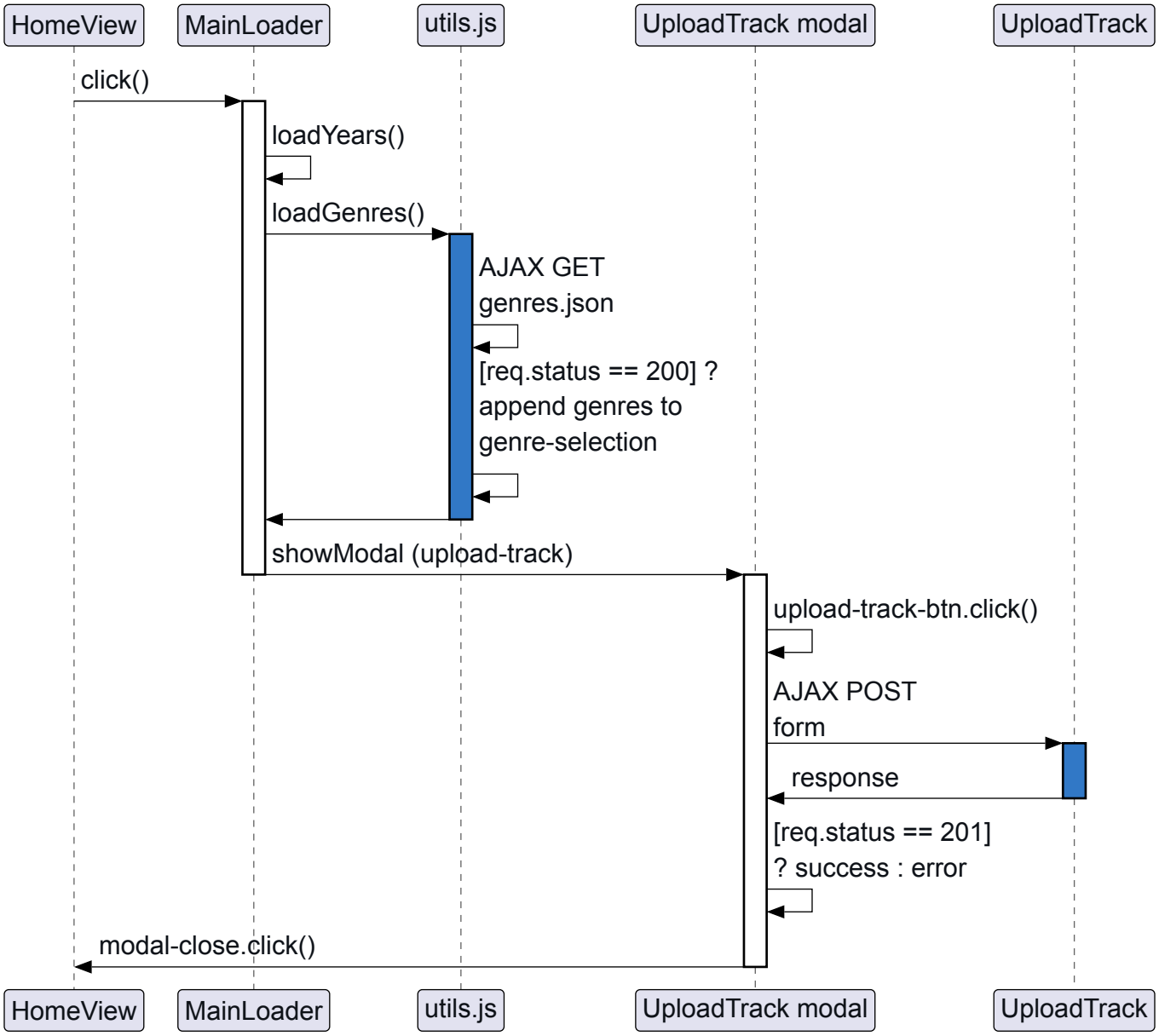
5.15 JS Evento: accesso Homepage



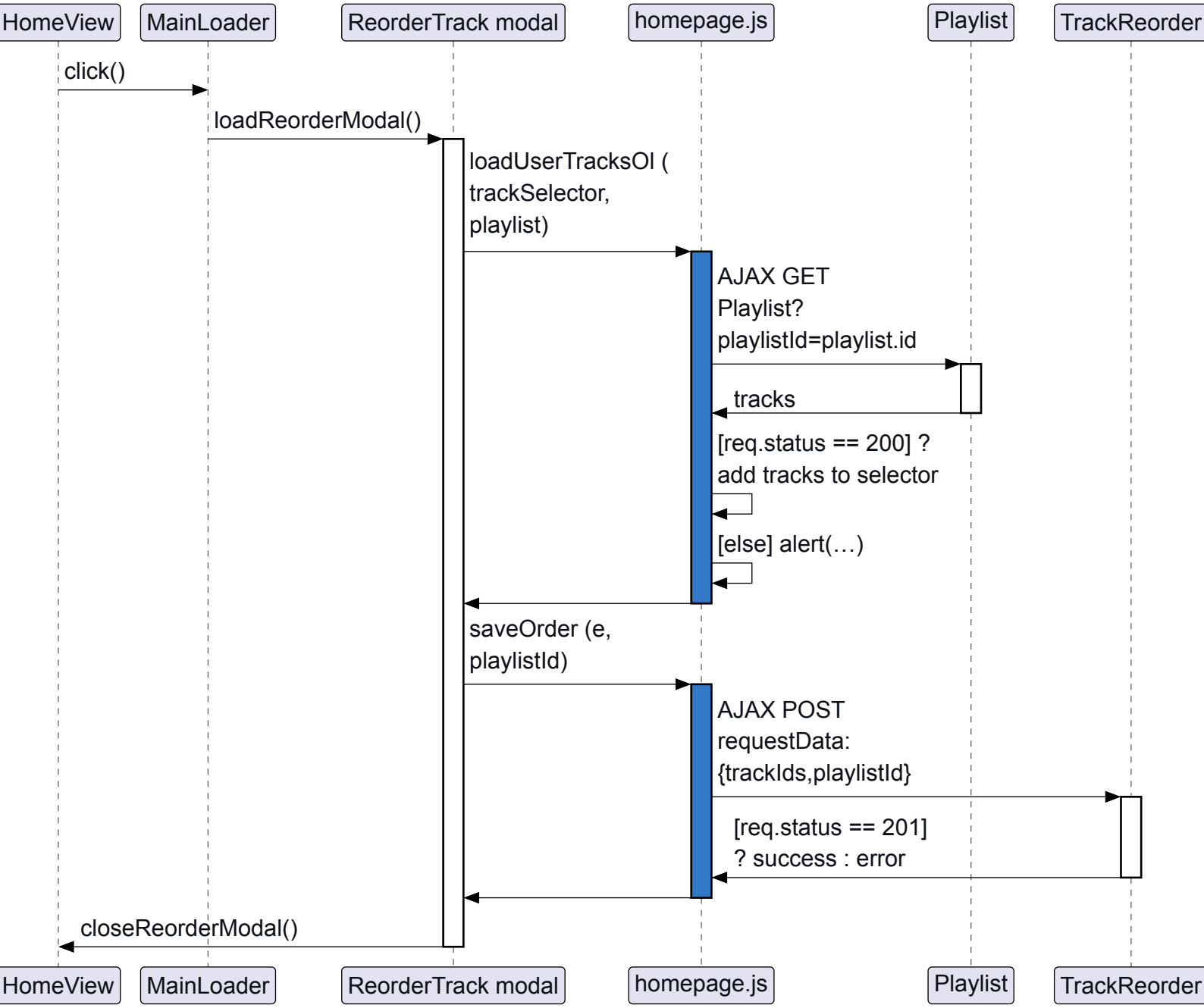
5.16 JS Evento: Accesso playlist



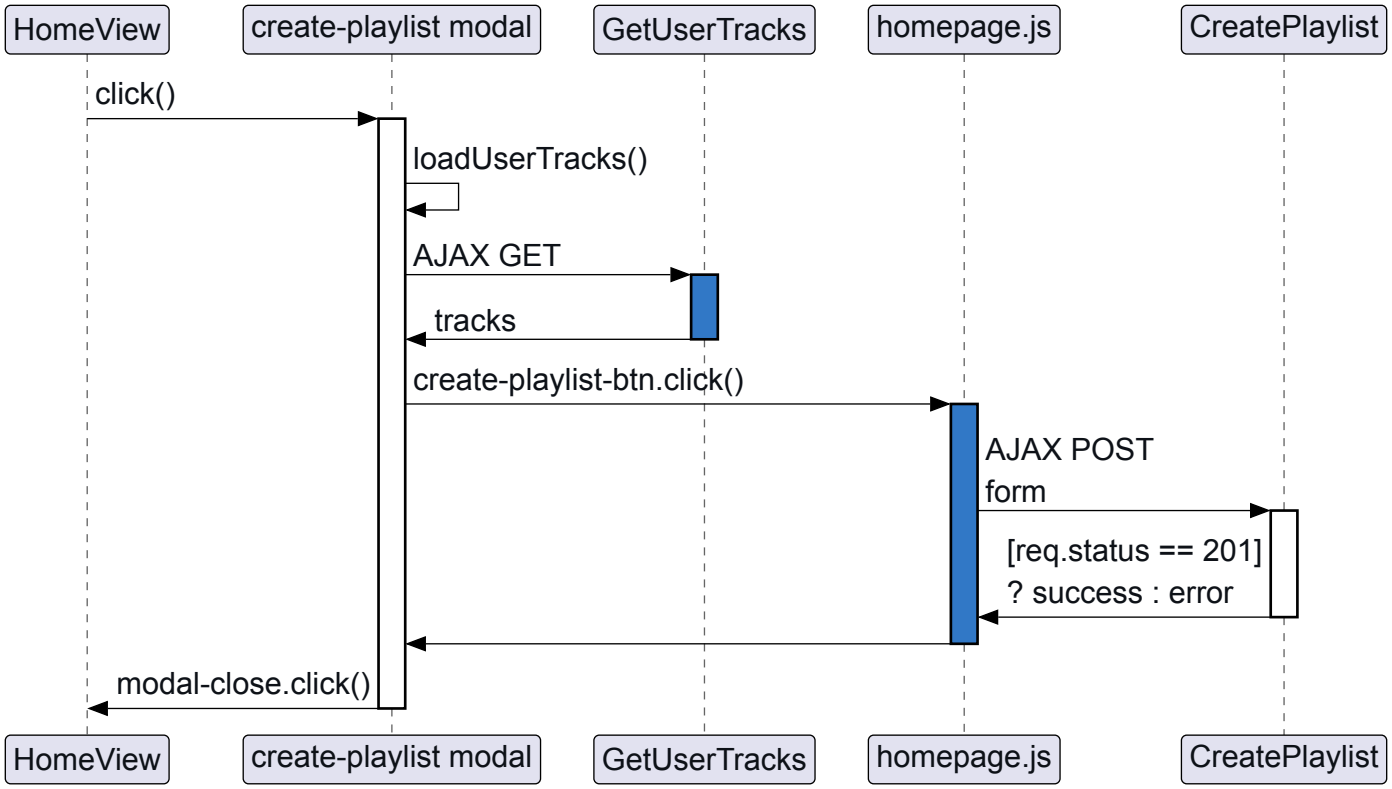
5.17 JS Evento: Upload Track modal



5.18 JS Evento: modale reorder



5.19 JS Evento: modale creazione Playlist



5.20 JS Evento: modale aggiungi traccia