Web Technologies Project @ PoliMi, 2025

Creating a Playlist Manager with Thymeleaf & TS

Manuel Zani
zani.manuel328@gmail.com
https://github.com/manuel3053

Contents

Abstract

Overview This project hosts the source code – which can be found on Github – for a web server that handles a playlist management system. A user is able to register, login and then upload tracks. The tracks are strictly associated to one user, similar to how a cloud service works. The user will be able to create playlists – sourcing from their tracks – and listen to them.

It should be noted there are two subprojects: a (pure) **HTML version**, which is structured as a series of separate webpages; and a **RIA version** (TS)[1], which is structured as a multi-page webapp. The functionalities are quite the same. For more information about the requirements for each version see Section 2.

Tools To create the project, the following technologies have been used:

- **Java**, for the backend server with servlets leveraging the **Springboot framework**
- **Typescript** for for the RIA one
- **Thymeleaf**, a template engine, for the HTML version
- **MariaDB** instead of MySQL, since it's an open source fork of MySQL

Credits

ringrazia vittorio

[1]For historic reasons, in the project is is referred as just `js`.

1

Original submission (in Italian)

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 RIA

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 personalizato 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

Project submission breakdown

2.1 Database logic

LEGEND		
	Entity	Attribute
	Attribute specification	Relationship

Each **user** has a **username**, **password**, **name** and **surname**. Each musical **track** is stored in the database via **title**, **image**, **album title**, **album artist name** (single or group), **album release year**, **musical genre** and **file**. Furthermore:

- Suppose the *genres are predetermined*
  `// the user cannot create new genres`
- It is not requested to store the track order within albums
- Suppose *each track can belong to a unique album* (no compilations)

After the login, the user is able to **create tracks** by loading their data and then group them in playlists. A **playlist** **is a set of chosen tracks** from the uploaded ones of the user. A playlist has a **title**, a **creation date** and is **associated to its creator**.

2.2 Behaviour

LEGEND		
	User action	Server action
	HTML page	Page element

After the login, the user **accesses** the **HOME PAGE** which **displays** the **list of their playlists**, ordered by descending creation date; a **form to load a track with relative data** and a **form to create a new playlist**. The playlist form:

- **Shows** the **list of user tracks** ordered by artist name in ascending alphabetic order and by ascending album release date
- The form allows to **select** one or more tracks

When a user **clicks** on a playlist in the **HOME PAGE**, the application **loads** the **PLAYLIST PAGE**; initially, it contains a **table with a row and five columns**:

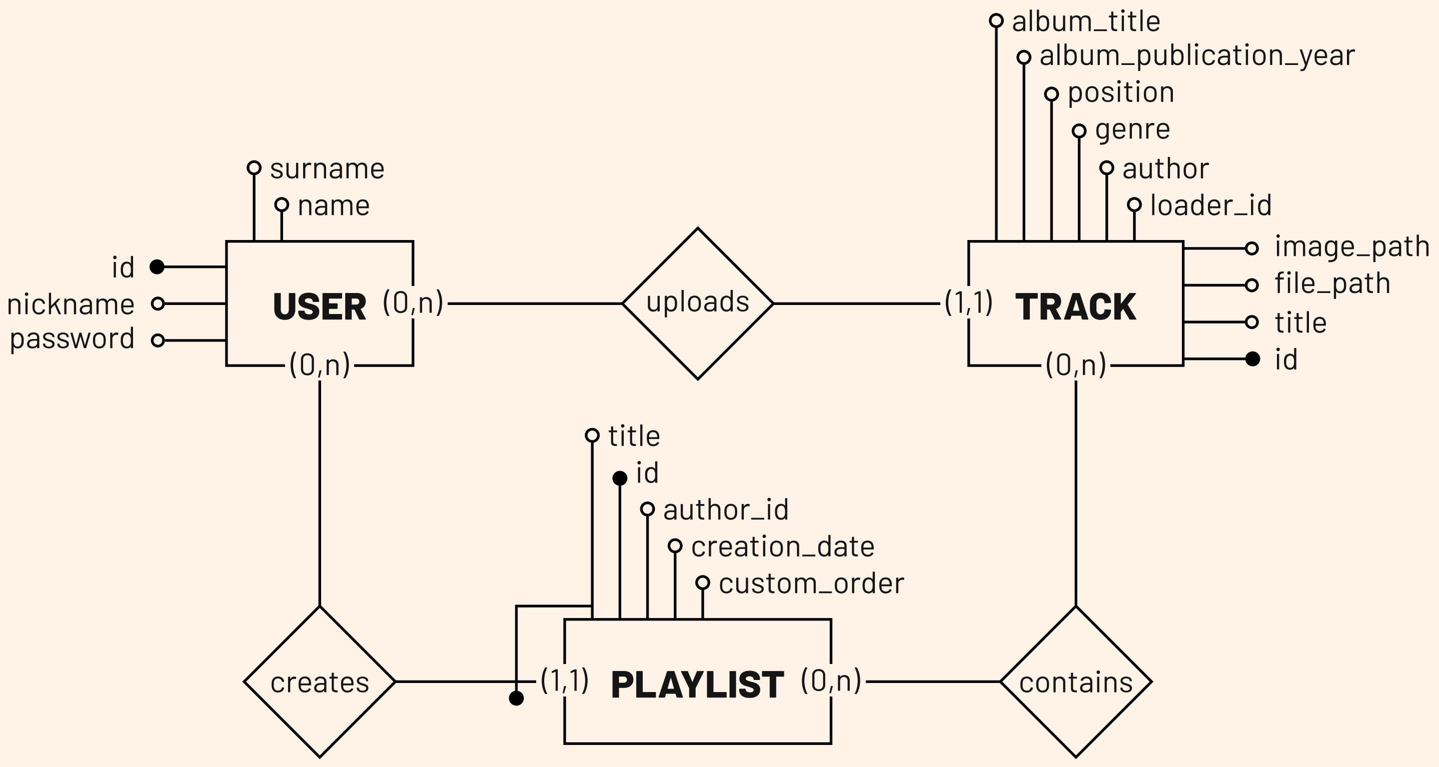


Figure 1: ER diagram (HTML and RIA).

- Every cell contains the track's title and album cover
- The tracks are ordered from left to right by artist name in ascending alphabetic order and by ascending album release date
- If a playlist contains more than 5 tracks, there are available commands to see the others (in blocks of five)

Playlist tracks navigation If the **PLAYLIST PAGE**:

1. Shows the first group and there are subsequent ones, a **NEXT button** appears on the right side of the row
2. Shows the last group and there are precedent ones, a **PREVIOUS button** appears on the left side of the row that allows to see the five preceding tracks
3. Shows a block of tracks and there are both subsequent and preceding ones, then on left and the right side appear both previous and next buttons

Track creation The **PLAYLIST PAGE** includes a **form that allows to add one or more tracks to the current playlist, if not already present**. This form acts in the same way as the playlist creation form.

After adding a new track to the current playlist, the application **refreshes the page** to display the first block of the playlist (the first 5 tracks). Once a user **selects the title of a track**, the **PLAYER PAGE** **shows** all of the **track data** and the **audio player**.

2.3 RIA version

Create a client-server web application that modifies the previous specification as follows:

- After the login, the entire application is built as a single webapp
- Every user interaction is managed without completely refreshing the page, but instead it asynchrounosly invokes the server and the content displayed is potentially updated
- The visualization event of the previous/next blocks is managed client-side without making a request to the server

Track reordering The application must allow the user to reorder the tracks in a playlist with a personalized order. From the **HOME PAGE** with an associated link to each playlist, the user **access** a modal window **REORDER** which shows the list of tracks ordered with the current criteria (custom or default).

The user can **drag** the title of a track and **drop** it in a different position to achieve the desidered order, without invoking the server. Once finished, the user can click on a **button to save the order** and **store** the sequence on the server. In subsequent accesses, the personalized track order is **loaded** instead of the default one. A newly added track in a custom-ordered playlist is **inserted always at the end**.

2.4 Added features

- Logout function (Section 5.3)
- Subscribe function (Section 5.4)

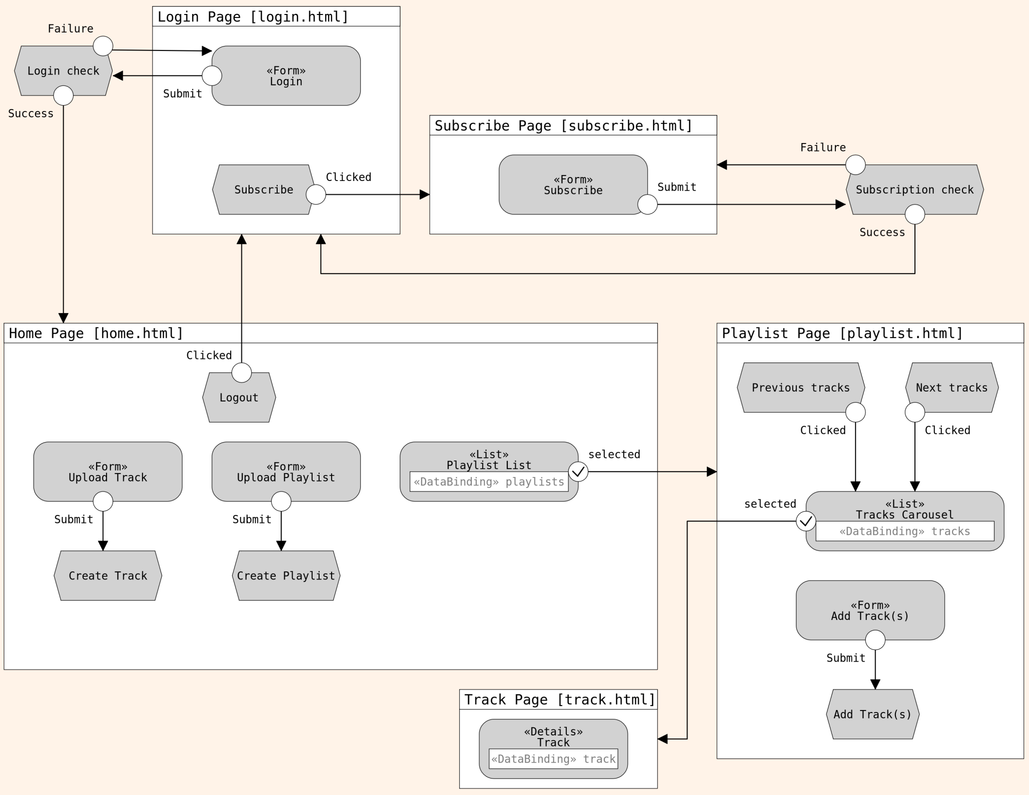


Figure 2: IFML diagram (HTML).

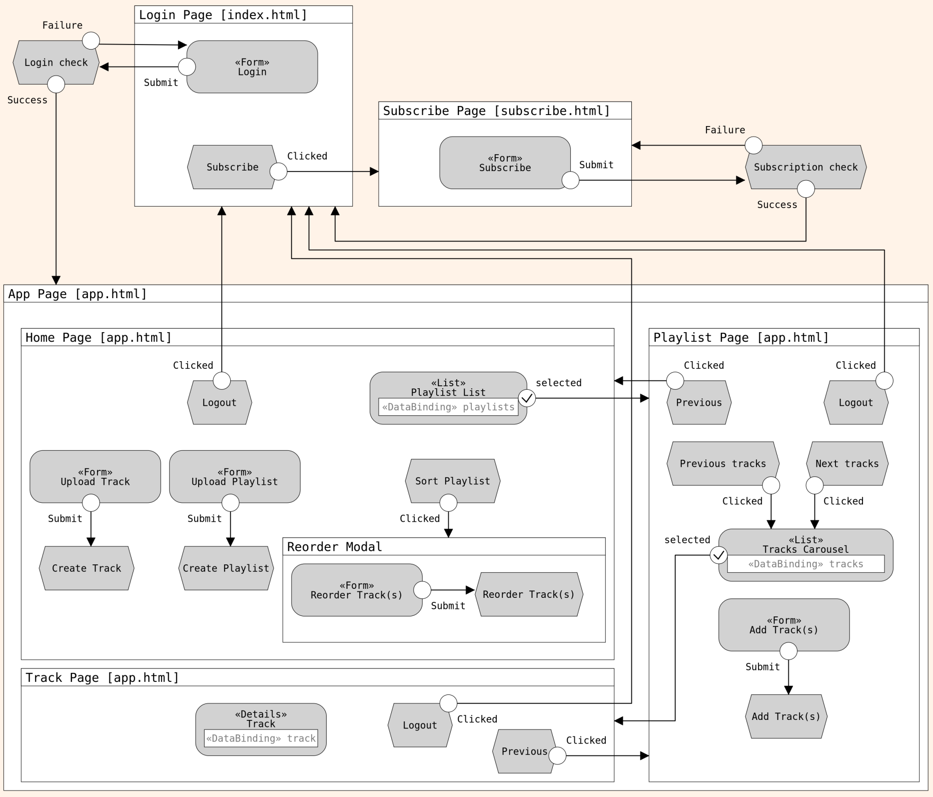


Figure 3: IFML diagram (RIA).

3

SQL database schema

3.1 Overview

The project requirements slightly change from pure_html and js, where the latter requires the playlist to support custom ordering the tracks – this is achieved by adding an optional position column in track and a custom order flag in playlist

In both scenarios, the schema is the same.

3.2 The tables

```sql
CREATE TABLE user
(
    id       integer AUTO_INCREMENT,
    username VARCHAR(64) not null,
    password VARCHAR(64) not null,
    name     VARCHAR(64) not null,
    surname  VARCHAR(64) not null,

    PRIMARY KEY (id),
    UNIQUE KEY `username` (`username`),
);
```

it is quite staightforward and standard. Apart from the id attribute, which is the primary key, the only other attribute that has a unique constraint is username.

```sql
CREATE TABLE track
(
    id          integer      AUTO_INCREMENT,
    loader_id   integer      not null ,
    file_path   VARCHAR(512) not null,
    image_path  VARCHAR(512) not null,
    title       VARCHAR(64)  not null,
    author      VARCHAR(64)  not null,
    album_title VARCHAR(64)  not null,
    album_publication_year integer not null,
    genre       VARCHAR(64)  not null,
    position    integer,

    PRIMARY KEY (id),
    UNIQUE (loader_id, title, author,
    album_title, album_publication_year),
    FOREIGN KEY (loader_id)
    REFERENCES user (id)
    ON UPDATE CASCADE ON DELETE CASCADE
);
```

the unique constraint on loader_id, title, author, album_title, album_publication_year is to make sure that the user doesn't load duplicates (there are almost all the attributes inside it to address the unlikely situation where an almost identical track is loaded).

The loader_id foreign key references the id of the user and if it is removed, his tracks are also removed.

```sql
CREATE TABLE playlist
(
    id            integer AUTO_INCREMENT,
    title         VARCHAR(64) not null,
    creation_date DATETIME not null DEFAULT
NOW(),
    author_id     integer     not null,
    custom_order  BOOL        not null,

    PRIMARY KEY (id),
    UNIQUE (title, author_id),
    FOREIGN KEY (author_id)
    REFERENCES user (id)
    ON UPDATE CASCADE ON DELETE CASCADE
);
```

The creatione_date attribute defaults to the today's date; and there is also the unique constraint on title, author_id because a playlist is bound to a single user (who can't have duplicate playlists – that is with the same title) via the foreign key.

```sql
CREATE TABLE playlist_tracks
(
    id          integer AUTO_INCREMENT,
    playlist_id integer not null,
    track_id    integer not null,

    PRIMARY KEY (id),
    FOREIGN KEY (playlist_id)
    REFERENCES playlist (id)
    ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (track_id)
    REFERENCES track (id)
    ON UPDATE CASCADE ON DELETE CASCADE
);
```

This table represents the "Contains" relation in the ER diagram (Figure 1). If a track or a playlist is removed their relationship is also removed.

4

Codebase overview

	HTML	*TS*
Entity	• Playlist • PlaylistTracks • Track • User	
DAO	• PlaylistDAO ‣ findByAuthorIdOrderByCreationDateAsc ‣ save ‣ findByAuthorIdAndId TS ‣ setCustomOrder TS • TrackDAO ‣ save ‣ findTrackByIdAndLoaderId ‣ getAllByUserIdSorted ‣ getPlaylistTracksGroup ‣ getAllNotInPlaylist ‣ getAllInPlaylist TS ‣ getAllInPlaylistCustom TS ‣ updatePosition TS	• PlaylistTracksDAO ‣ save ‣ getAllByPlaylistId • UserDAO ‣ save ‣ findByUsername
Controller	• FileController ‣ serveSafeFile • HomeController ‣ showPage ‣ addTrack ‣ addPlaylist • LoginController ‣ showPage • PlaylistConstroller ‣ showPage ‣ addTrackToPlaylist • SubscribeController ‣ showPage ‣ subscribe • TrackController ‣ showPage	• FileController ‣ serveSafeFile • AuthController ‣ login ‣ subscribe • PlaylistController ‣ getPlaylists ‣ getPlaylistSizeByid ‣ addPlaylist ‣ addTracksToPlaylist ‣ setCustomOrder • TrackConstroller ‣ getTrackById ‣ getTracks ‣ getAllNotInPlaylist ‣ getAllInPlaylist ‣ getGenres ‣ addTrack
Springboot Config	• UserService ‣ loadUserByUsername • UserWithId • SecurityConfig ‣ securityFilterChain	• SecurityConfig ‣ securityFilterChain ‣ securityContextRepository ‣ authenticationManager

Table 1: Components comparison

Client side		Server side	
Event	Action	Event	Action
Index ⇒ Login form ⇒ Submit	Data validation	POST (`username`, `password`)	Credentials check
HomeView ⇒ Load	Loads all User playlists	GET (`user playlists`)	Queries user playlists
HomeView ⇒ Click on a playlist	Loads all tracks associated to that Playlist	GET (`playlistId`)	Queries the tracks associated to the given playlistId
HomeView ⇒ Click on reorder button	Load a modal to custom order the track in the Playlist	GET (`playlistId`)	Queries the tracks associated to the given playlistId
Reorder modal ⇒ Save order button	Saves the custom order to the database	POST (`trackIds`, `playlistId`)	Updates the `playlist_tracks` table with the new custom order
Create playlist modal ⇒ Create playlist button	Loads the modal to create a new playlist; returns the newly created playlist if successful	POST (`playlistTitle`, `selectedTracks`)	Inserts the new Playlist in the `playlist` table
Upload track modal ⇒ Upload track button	Loads the modal to upload a new track; returns the newly uploaded track if successful	POST (`title`, `artist`, `year`, `album`, `genre`, `image`, `musicTrack`)	Inserts the new Track in the `tracks` table
Sidebar ⇒ Playlist button	Views the last selected Playlist, if one had been selected	GET (`last selected Playlist`)	Queries the tracks associated to the given playlistId
Sidebar ⇒ Track button	Views the last selected Track, if one had been selected	GET (`last selected Track`)	Queries the data associated with the given trackId
Sidebar ⇒ HomePage	Returns to the HomeView	GET (user playlists)	Queries user playlists
Logout	Invalidates the current User session	GET	Session invalidation

Table 2: Events & Actions.

Client side		Server side	
Event	Controller	Event	Controller
Index ⇒ Login form ⇒ Submit	`makeCall()` function	POST (`username, password`)	Login (servlet)
HomeView ⇒ Load	`HomeView.show()` (its invocation is done by the `MainLoader` class)	GET	Homepage (servlet)
HomeView ⇒ Click on a playlist	`loadPlaylist()`	GET (`playlistId`)	Playlist (servlet)
HomeView ⇒ Click on reorder button	`loadReorderModal()`	GET (`playlistId`)	Playlist (servlet)
Reorder modal ⇒ Save order button	`saveOrder()`	POST (`trackIds, playlistId`)	TrackReorder (servlet)
Create playlist modal ⇒ Create playlist button	`makeCall()`	POST (`playlistTitle, selectedTracks`)	CreateNewPlaylist (servlet)
Upload track modal ⇒ Upload track button	`makeCall()`	POST (`title, artist, year, album, genre, image, musicTrack`)	UploadTrack (servlet)
Sidebar ⇒ Playlist button	`playlistView.show()`	GET (`last selected Playlist`)	Playlist (servlet)
Sidebar ⇒ Track button	`trackView.show()`	GET (`last selected Track`)	Track (servlet)
Sidebar ⇒ HomePage	`homeView.show()`	GET (`user playlists`)	Homepage (servlet)
Logout	`makeCall()` function	GET	Logout (servlet)

Table 3: Events & Controllers (or event handlers).

4.1 Components

Introduction Both versions of the project present a lot of similarities but fot the TS version have been reorganized and some features were added

4.2 Backend

FAI IN MODO CHE LA TABELLA SIA POSIZIONATA PER IL BACKEND

4.3 Frontend

parla delle viste html e delle viste per typescript

4.4 RIA subproject

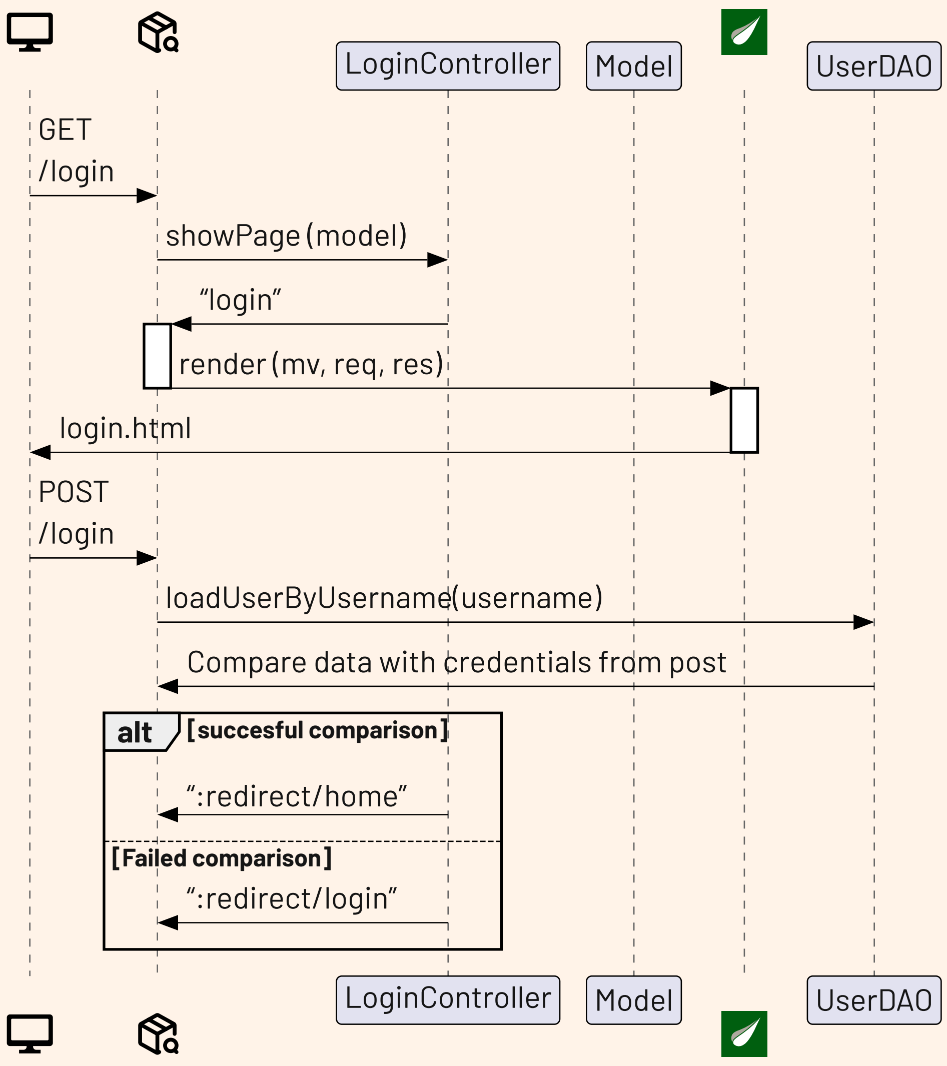
5

Sequence diagrams HTML

5.1 Disclaimer

- scrivi cosa è DispatcherServlet e associalo all'icona
- associa client ad icona
- spiega che tutti i redirect seguono lo stesso sequence di quando si richiede una pagina normalmente e che questo è stato omesso per semplicità
- Aggiungi che Springboot modifica molto i sequence diagram rispetto a delle basilari servlet
- Mostra le configurazioni di springboot con il codice e spiega perché servono in questa parte
- Dai una spiegazione unica sul flusso di presentazione di una pagina, così non devi farlo ogni volta
- Aggiungi che le eccezioni sono gestite ma non in modo user-friendly, dato che non era richiesto di gestire finemente le eccezioni ma solo di gestirle; inoltre il tempo a disposizione era poco
- Aggiungi che ogni volta che compare userId vuol dire che è stato chiamato UserDetailsExtractor che recupera l'id dell'utente dal SecurityContext

5.2 Login sequence diagram



Comment

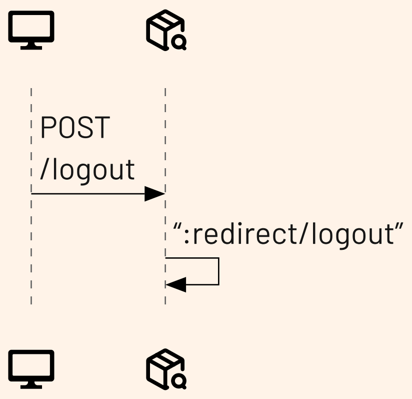
Once the server is up and running, the Client requests the Login page. Then, thymeleaf processes the request and returns the correct context to index the correct locale. Afterwards, the User inserts their credentials.

Those values are passed to the DispatcherServlet which sends the message to a POST /login mapping offered by Springboot: here the message is analyzed by different filters (almost all of them are used internally by Springboot), till it reaches the AuthorizationFilter.

The AuthorizationFilter calls internally the UserDetailsService Bean wich is implemented by the UserService class which, inside loadUserByUsername, queries the DB to receive a user associated to that username (that's why the UserDAO is called).

After this operation, the AuthorizationFilter filter compares the data provided by the client and by the UserDetailsService and if they are the same, the user is redirected to the home page. Otherwise the user is redirected to the login page.

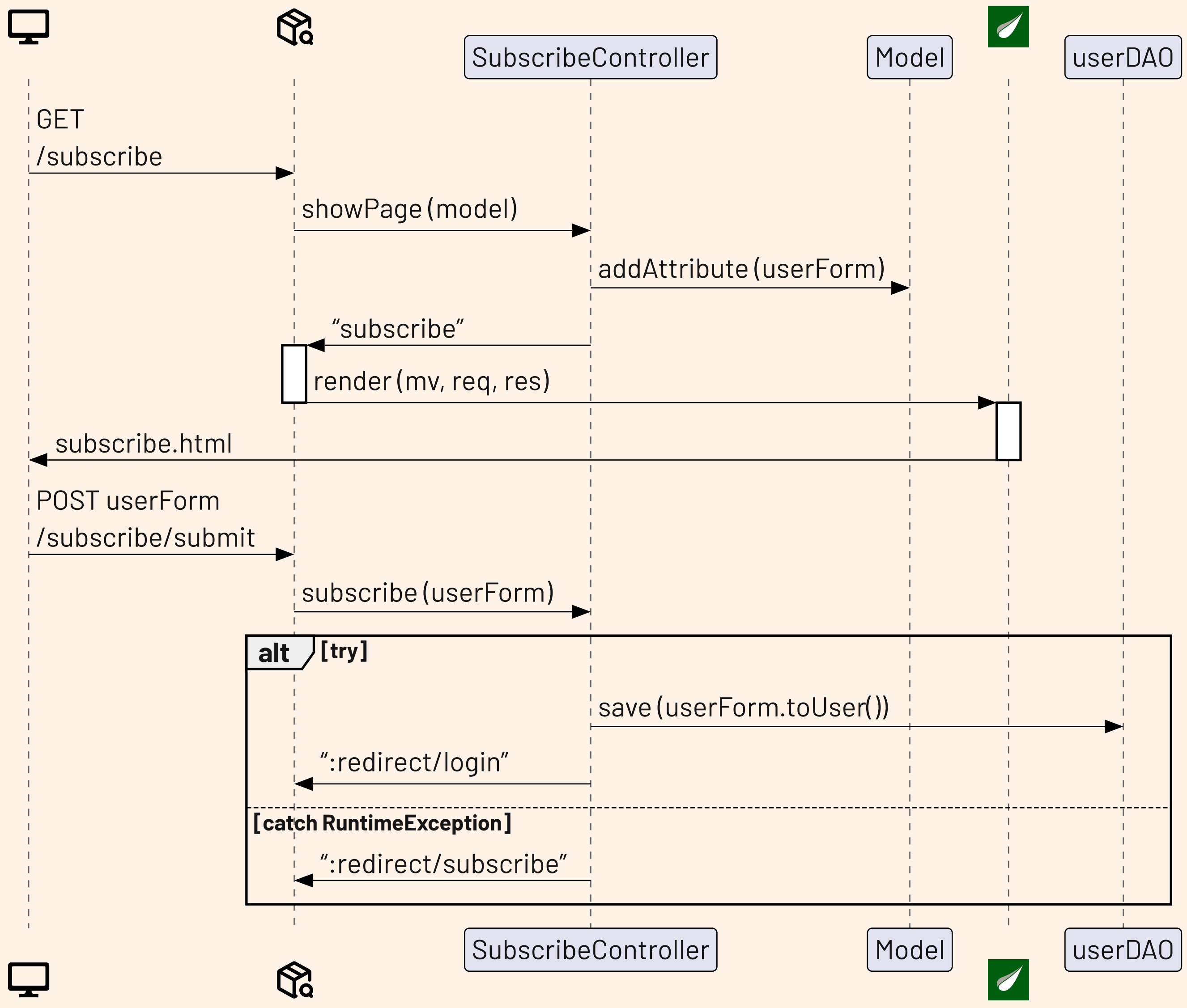
5.3 Logout sequence diagram



Comment

In order to perform a logout, Springboot requires the client to call the POST /logout mapping which internally invalidates session and all the other data associated to the user (of course the data inside the DB are preserved), and then the user is redirected to the login page.

5.4 Subscribe sequence diagram



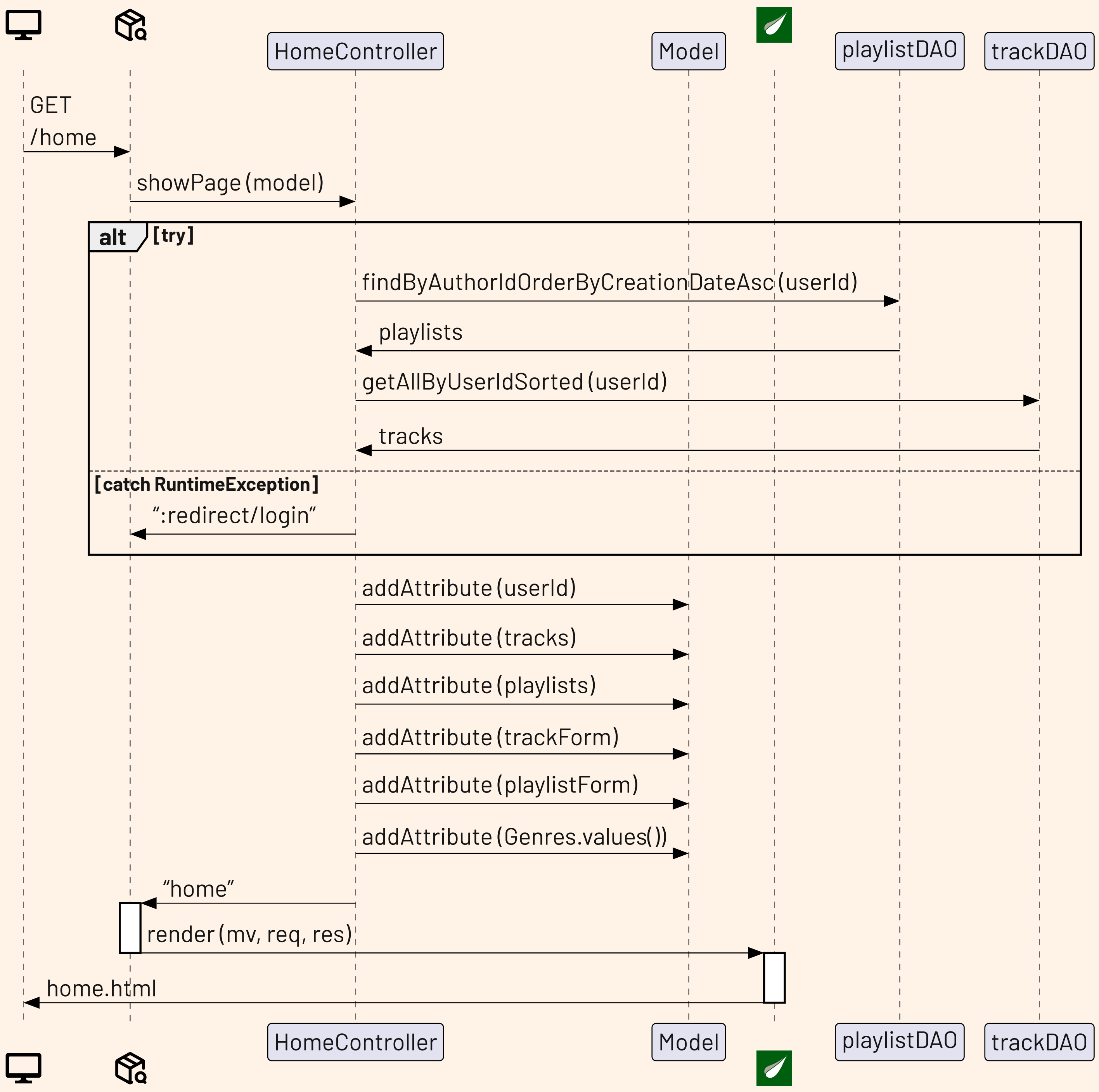
Comment

After requesting the page with GET /subscribe, the user can fill and submit the provided form to subscribe to the site.

The call is redirected to the correct controller (SubscribeController) by the DispatcherServlet: the controller tries to save the user by calling userDAO.save() and if any kind of RuntimeException exceptio occurs (so SQLExceptions are also considered), the user is redirected to the subscribe page. Otherwise the user is redirected to the login page.

In this situation an SQLExceptions exception might occour also because in the DB there is already a user with the same username.

5.5 Show home page sequence diagram



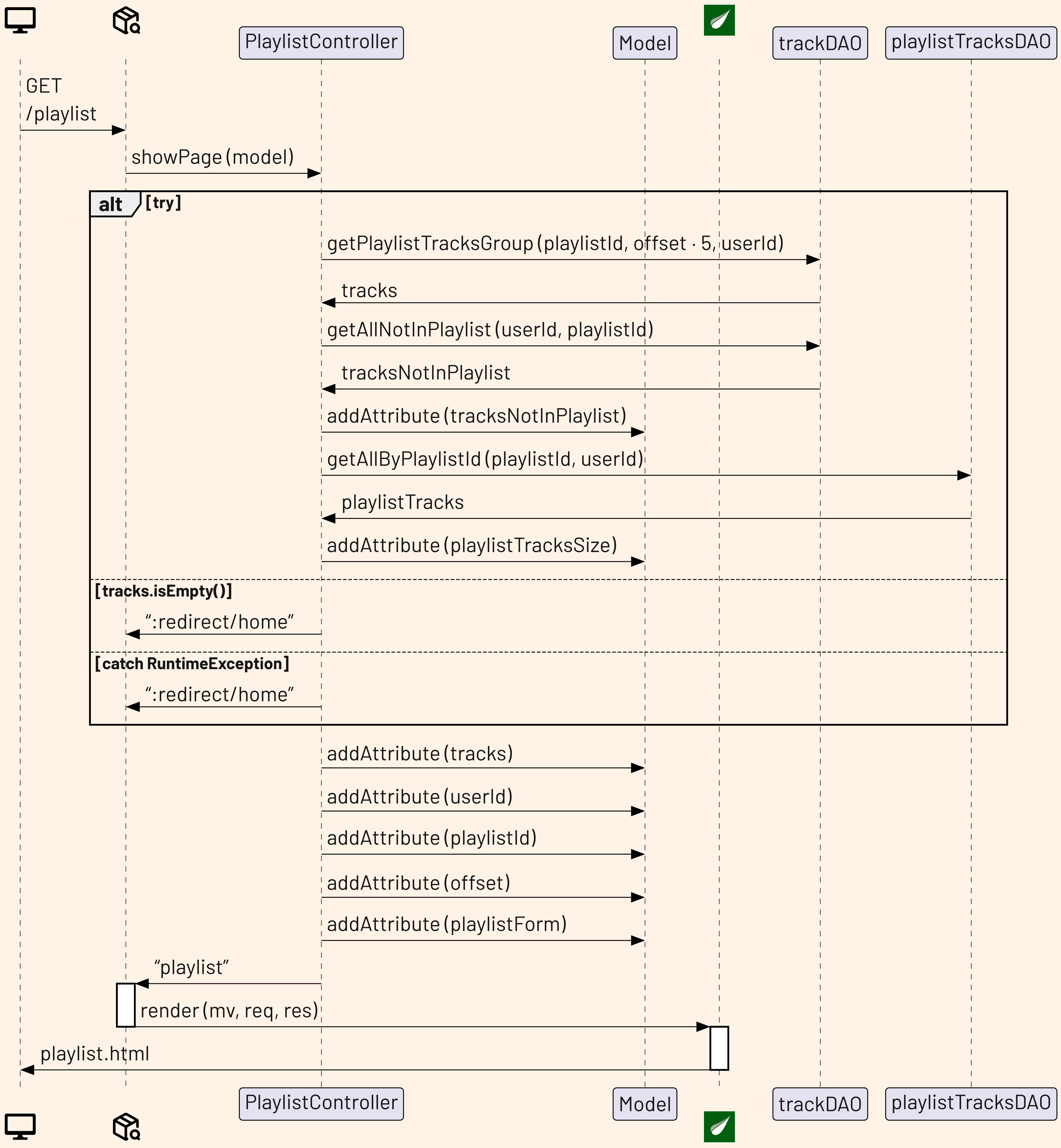
Comment

When the user requests the home page, the HomeController obtains the user id and requests: all playlists owned by the user, sorted as pointed in the submission; all tracks owned by the user, sorted as pointed in the submission The results of these requests are added inside the thymeleaf Model.

If a RuntimeException occours the user is redirected to the login page.

Inside the Model are also added the trackForm and Genre.values() to manage the upload of a track and playlistForm to manage the upload of a playlist

5.6 Show playlist page sequence diagram



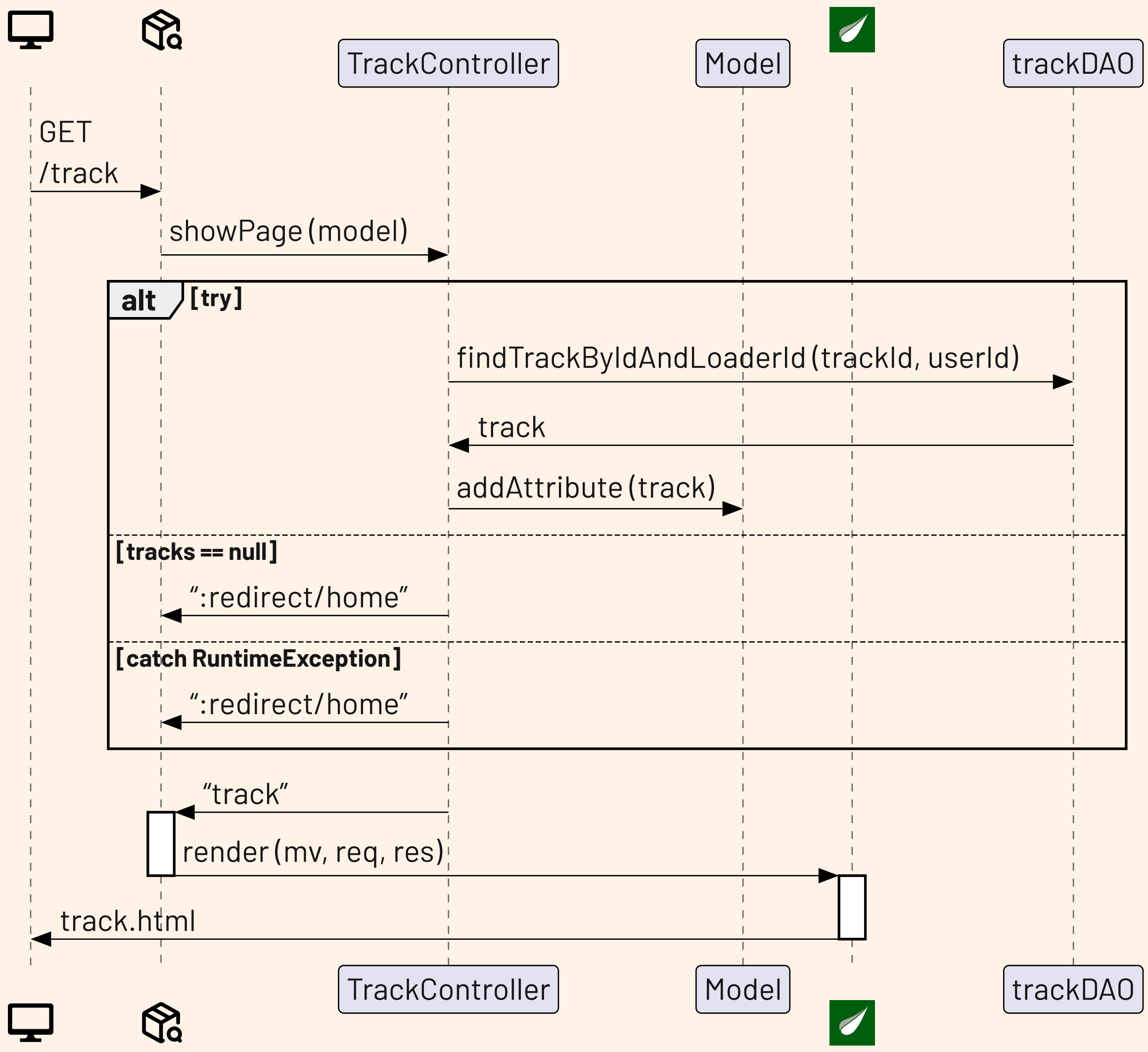
Comment

When the user requests the playlist page, the PlaylistController obtains the user id and requests: the first group of tracks in the playlist; all tracks not in the playlist; all the tracks in the playlist The results of these requests are added inside the thymeleaf Model (for the last one only the size is considered).

If a RuntimeException occours or the playlist is empty, the user is redirected to the login page.

Inside the Model is also added playlistForm to manage the insertion of more tracks in the playlist.

5.7 Show track page sequence diagram

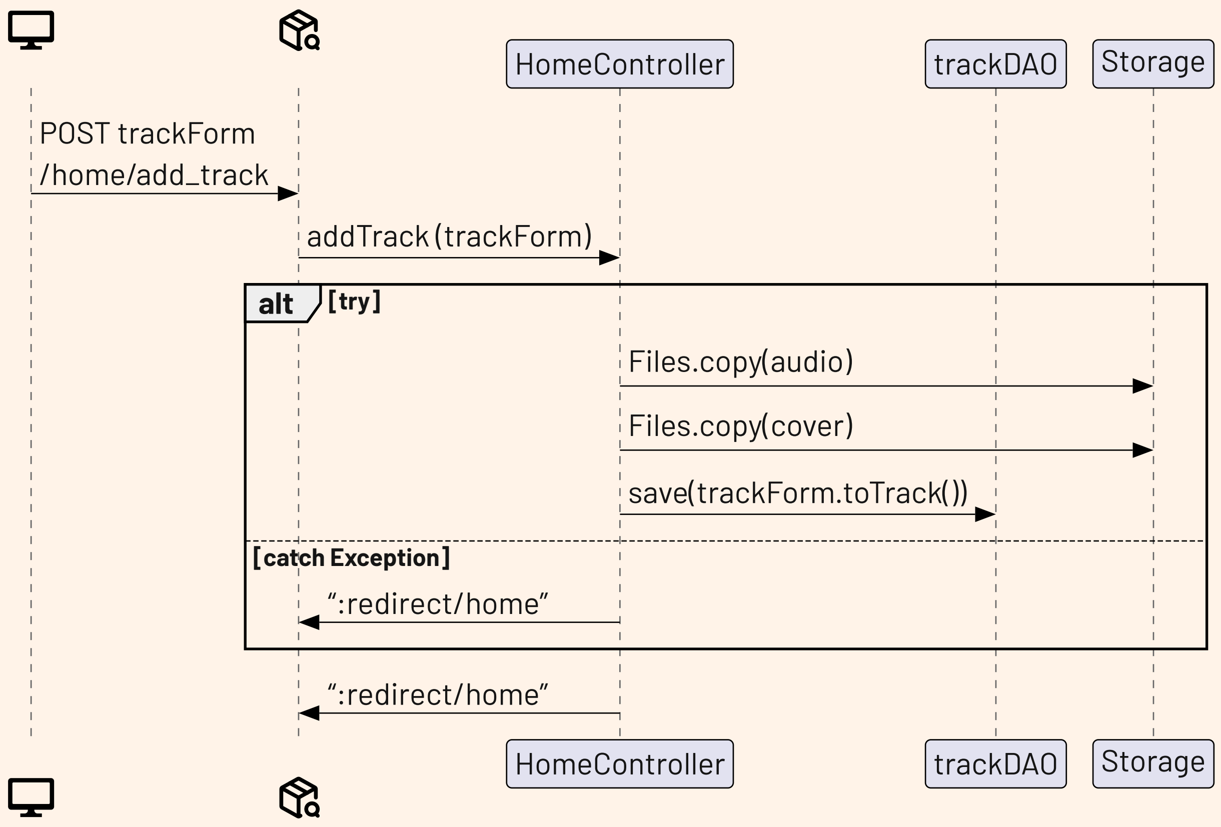


Comment

When the user requests the track page, the TrackController obtains the user id and requests the track that the user wants to display. The result is added to the model to display the data about the track.

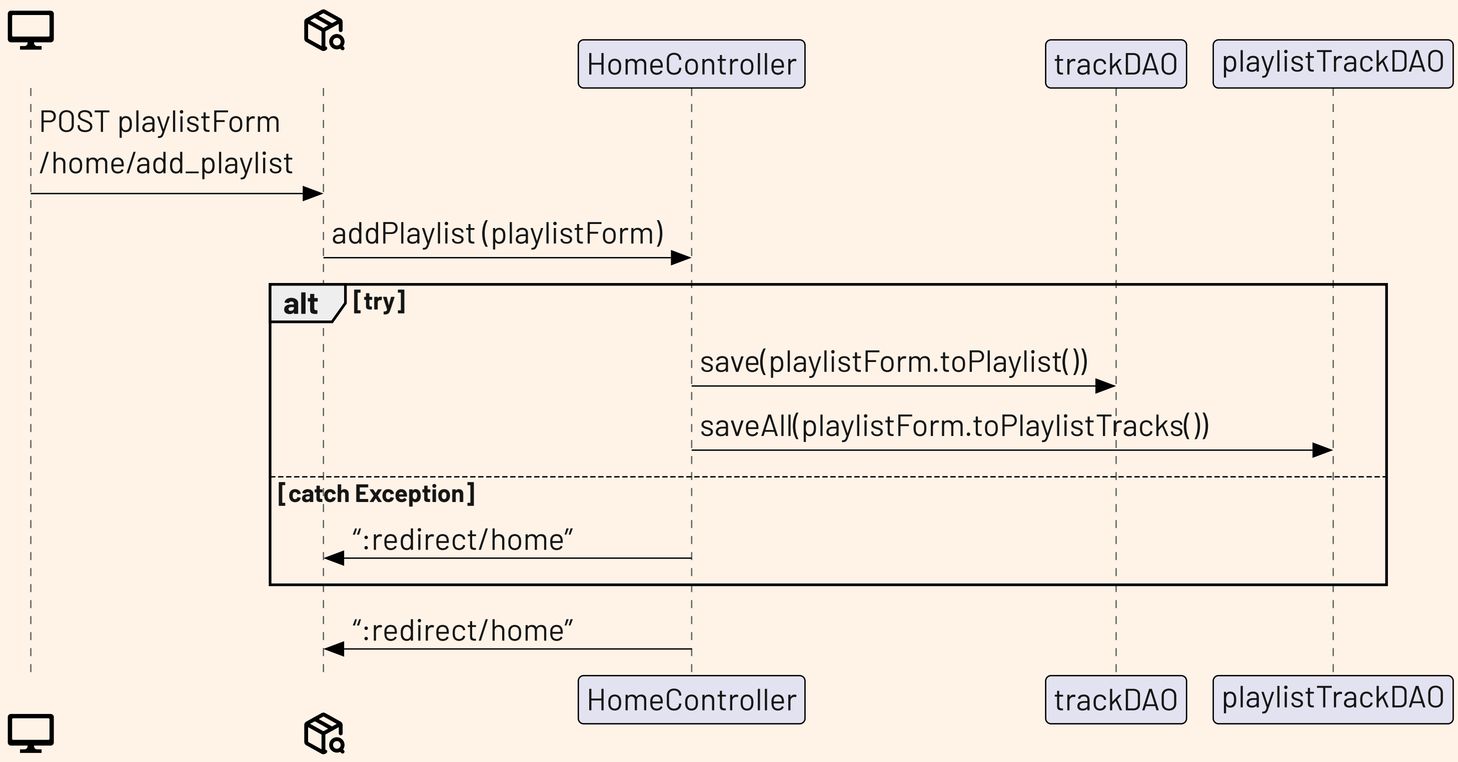
If a RuntimeException occours or the track is null, the user is redirected to the home page.

5.8 Add track sequence diagram



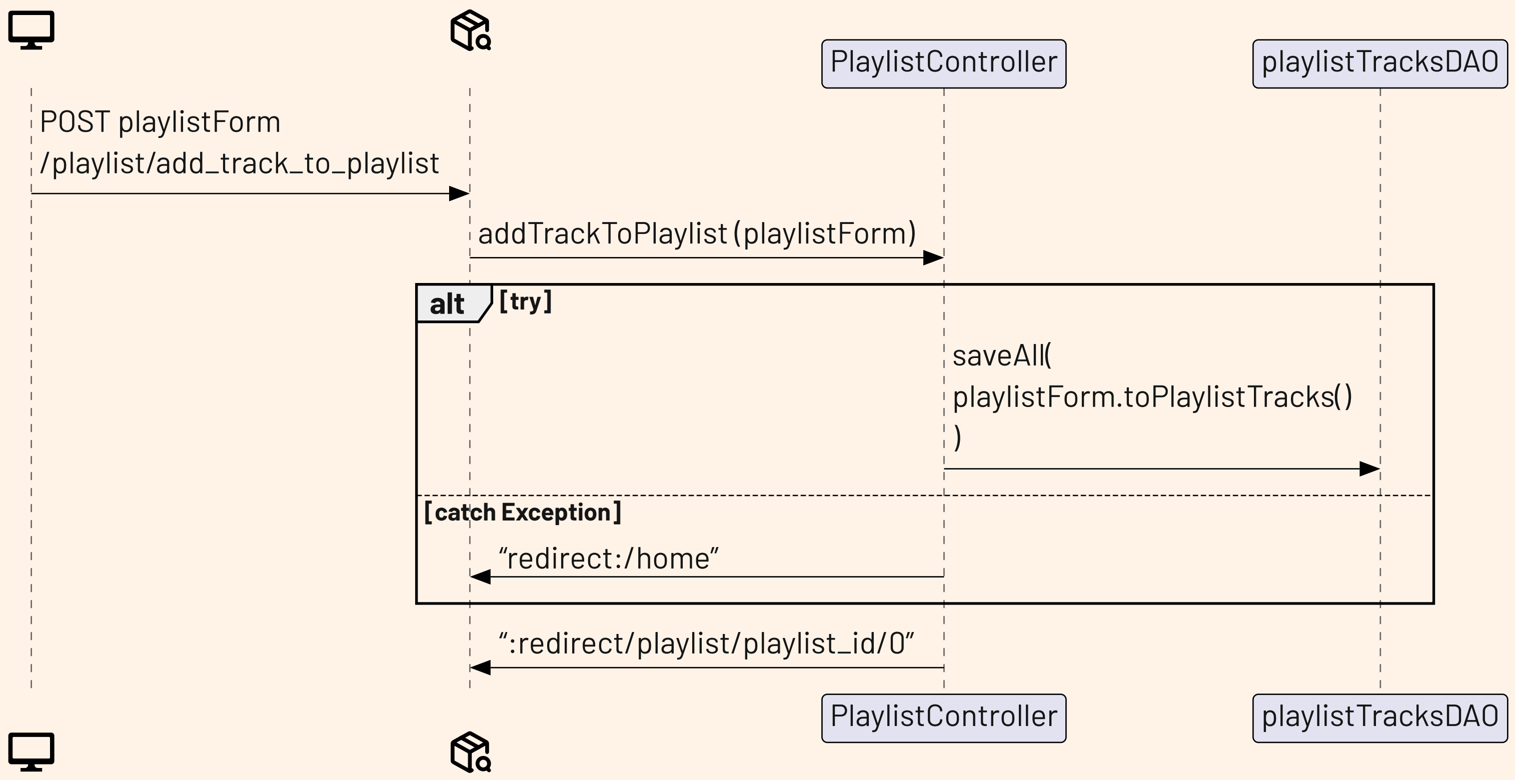
Comment

5.9 Add playlist sequence diagram



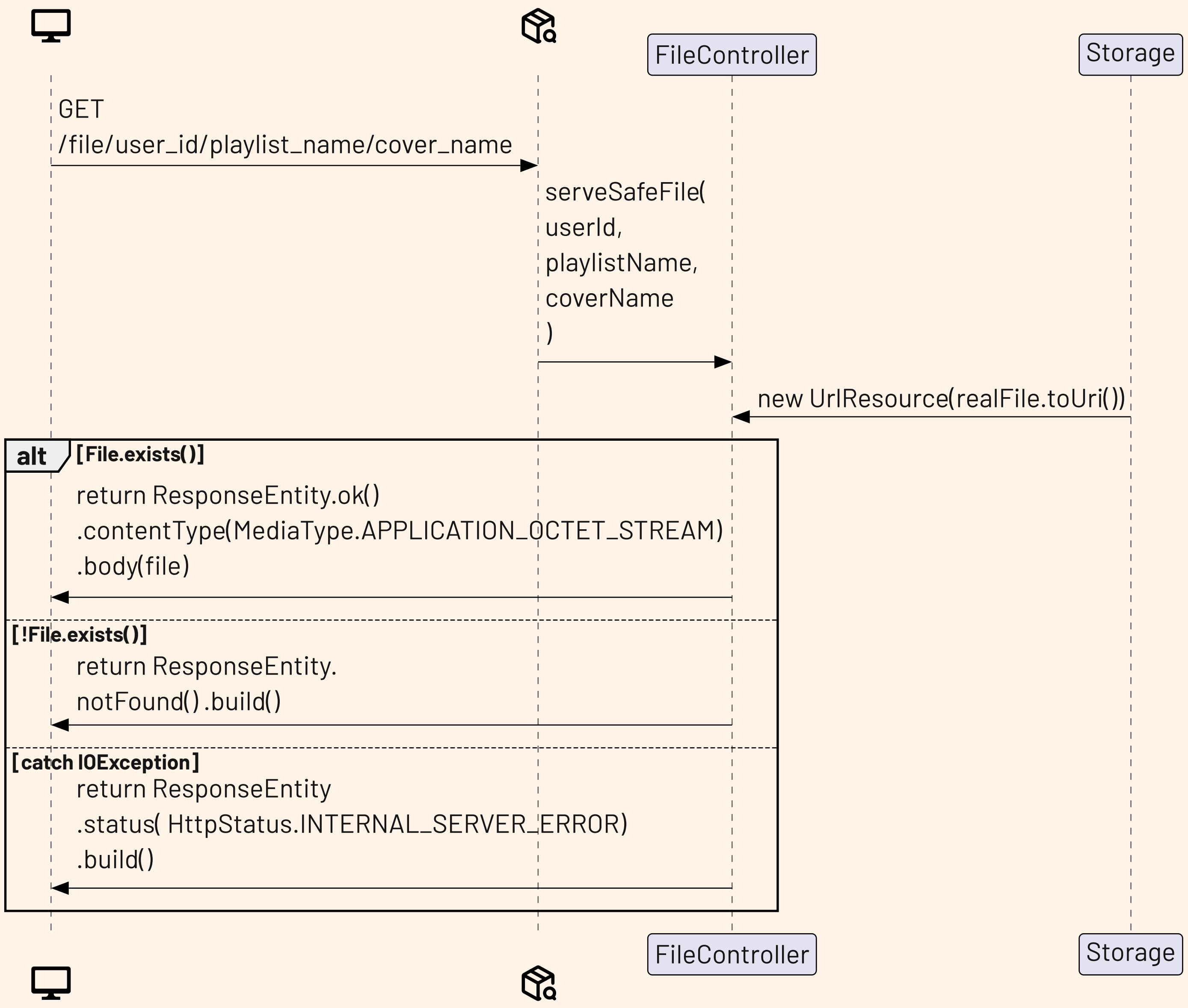
Comment

5.10 Add track to playlist sequence diagram



Comment

5.11 Get file sequence diagram

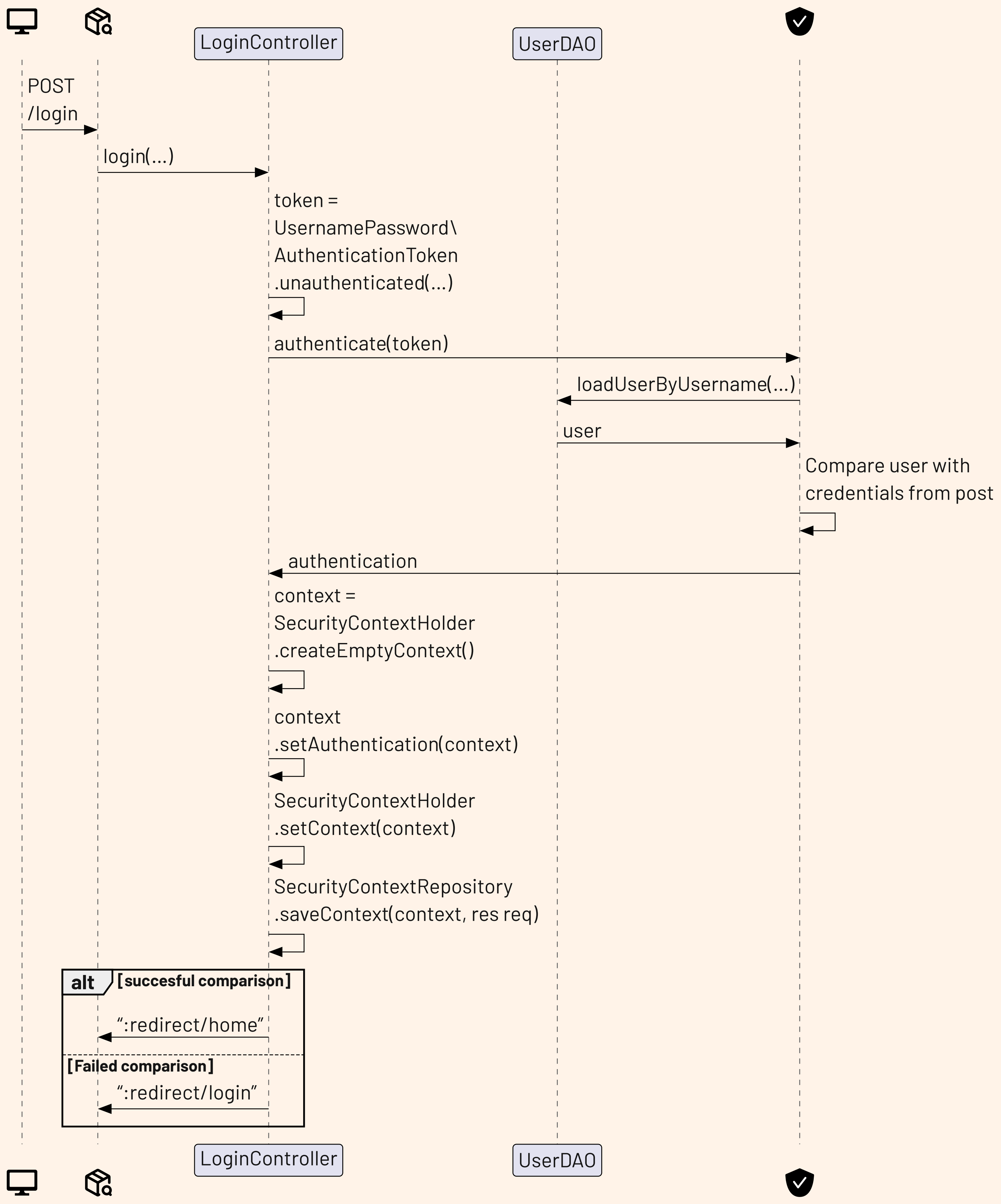


Comment

6

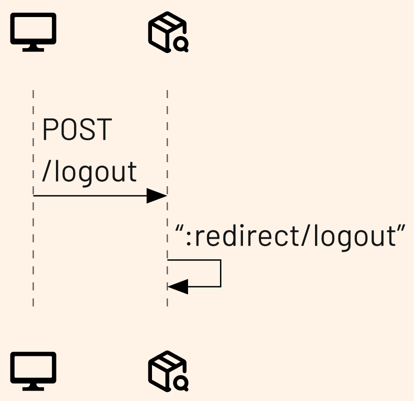
Sequence diagrams RIA

6.1 Login sequence diagram


LoginController
UserDAO
POST /login
login(...)
token = UsernamePassword\ AuthenticationToken .unauthenticated(...)
authenticate(token)
loadUserByUsername(...)
user
Compare user with credentials from post
authentication
context = SecurityContextHolder .createEmptyContext()
context .setAuthentication(context)
SecurityContextHolder .setContext(context)
SecurityContextRepository .saveContext(context, res req)
alt
[succesful comparison]
":redirect/home"
[Failed comparison]
":redirect/login"
LoginController
UserDAO

Comment

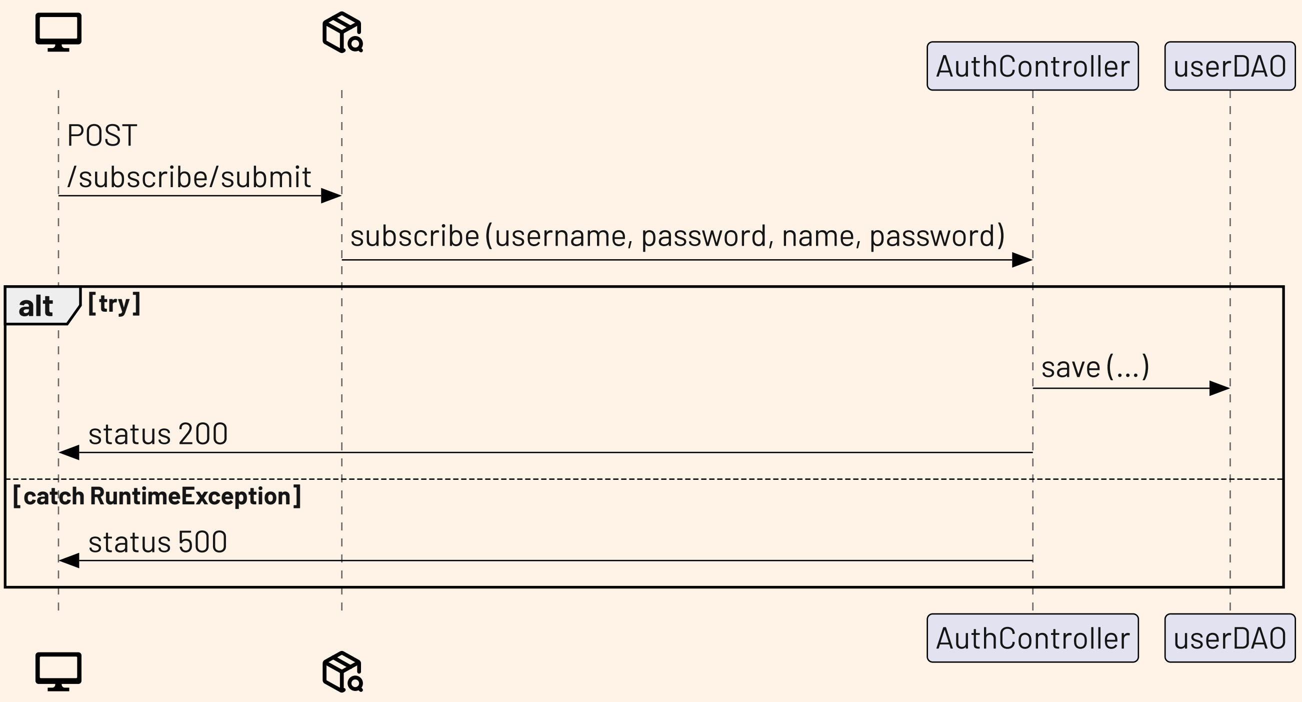
6.2 Logout sequence diagram



Comment

In order to perform a logout, Springboot requires the client to call the POST /logout mapping which internally invalidates session and all the other data associated to the user (of course the data inside the DB are preserved), and then the user is redirected to the login page.

6.3 Subscribe sequence diagram



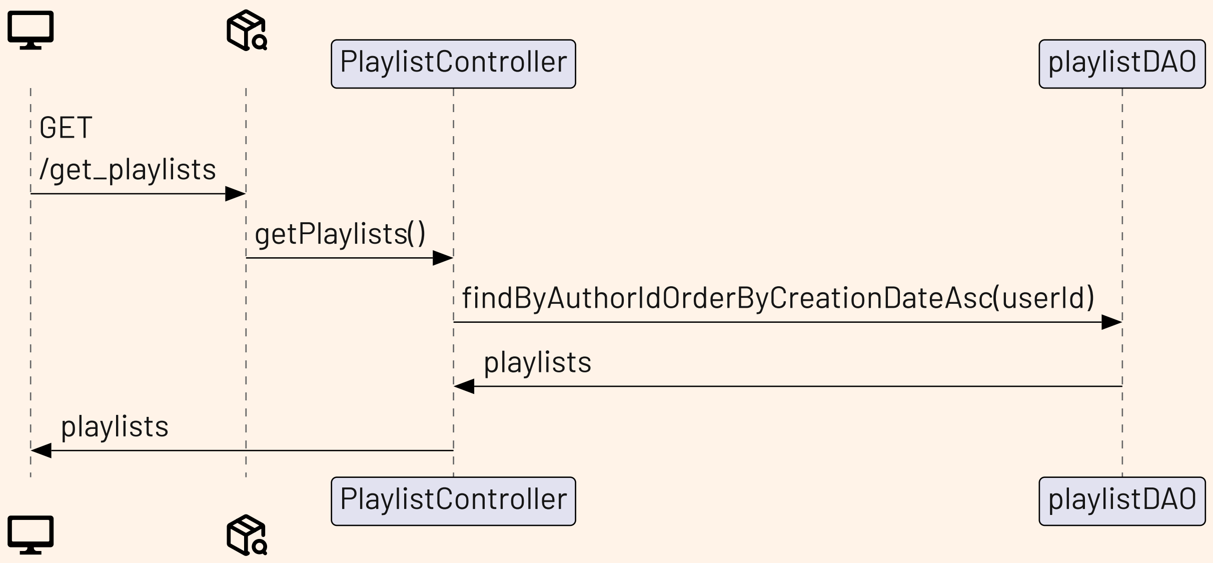
Comment

After requesting the page with GET /subscribe, the user can fill and submit the provided form to subscribe to the site.

The call is redirected to the correct controller (SubscribeController) by the DispatcherServlet: the controller tries to save the user by calling userDAO.save() and if any kind of RuntimeException exceptio occurs (so SQLExceptions are also considered), the user is redirected to the subscribe page. Otherwise the user is redirected to the login page.

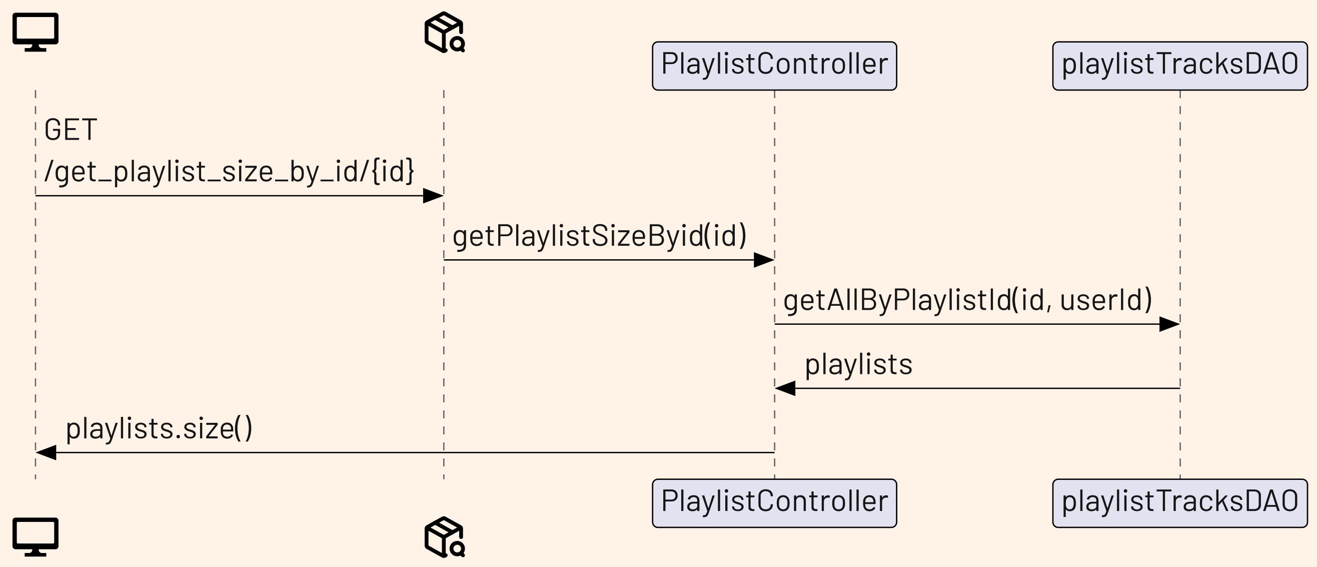
In this situation an SQLExceptions exception might occour also because in the DB there is already a user with the same username.

6.4 GetPlaylist sequence diagram



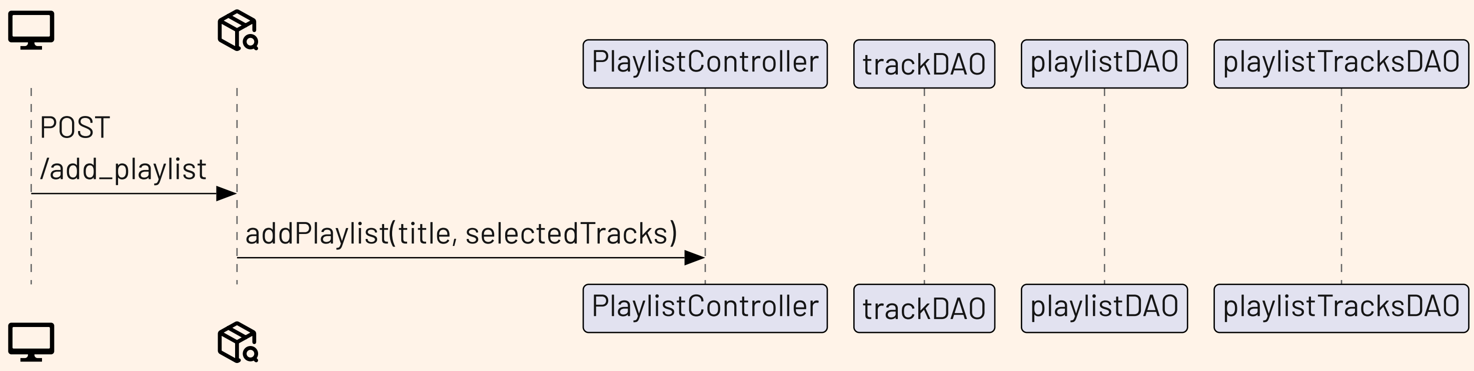
Comment

6.5 GetPlaylistSizeByid sequence diagram



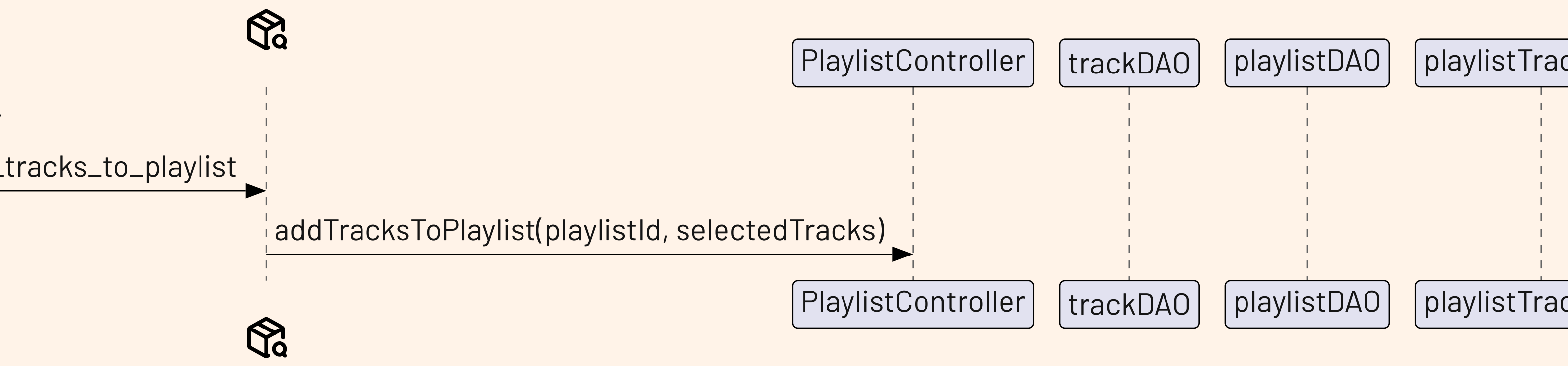
Comment

6.6 AddPlaylist sequence diagram



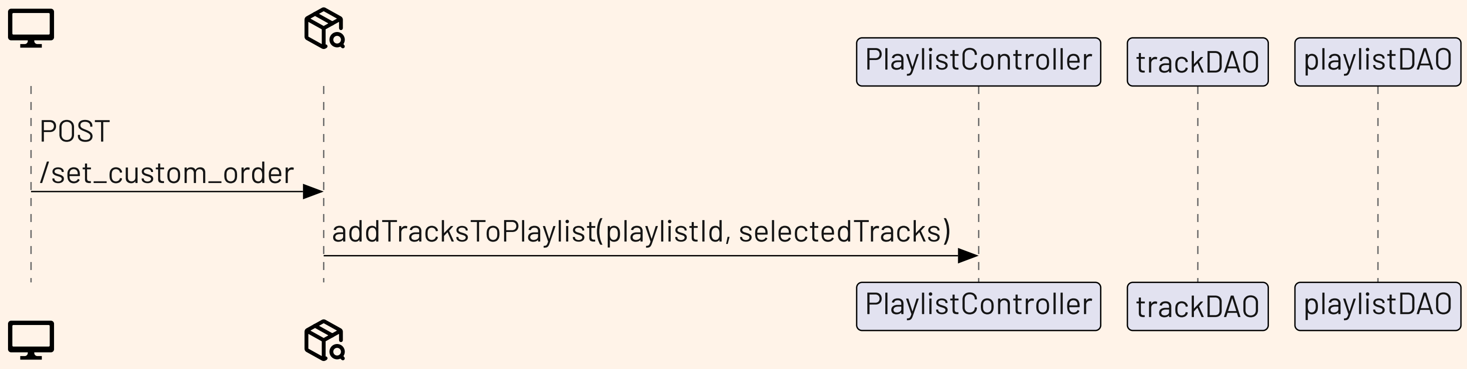
Comment

6.7 AddTracksPlaylist sequence diagram



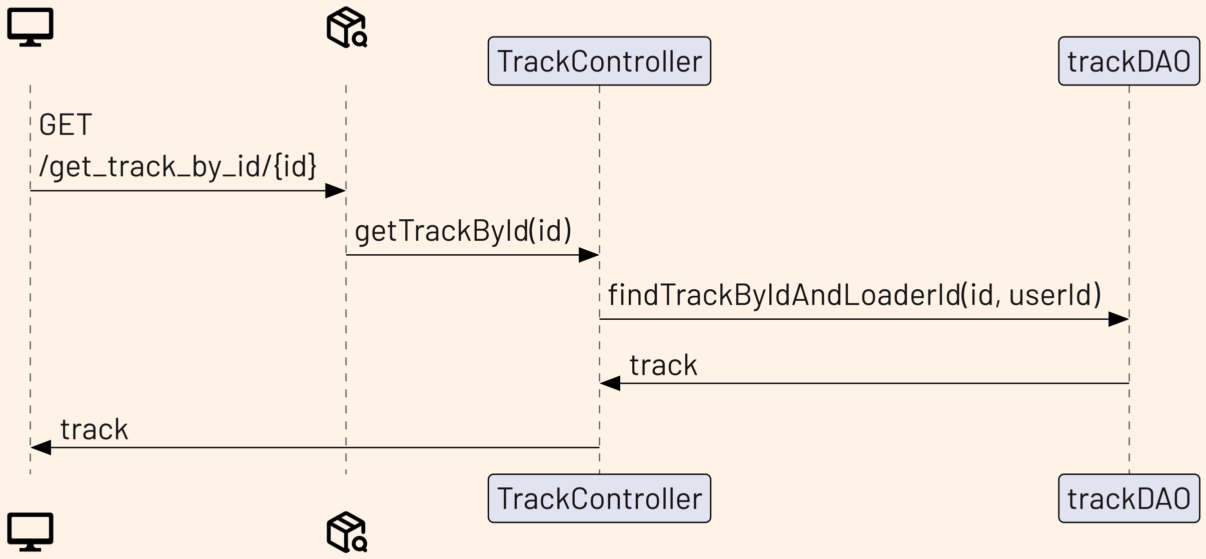
Comment

6.8 SetCustomOrder sequence diagram



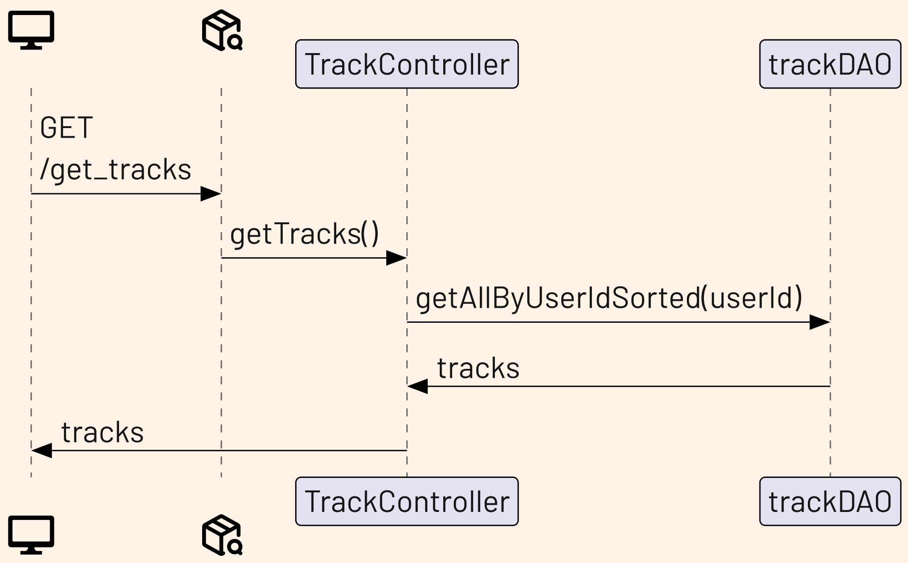
Comment

6.9 getTrackById sequence diagram



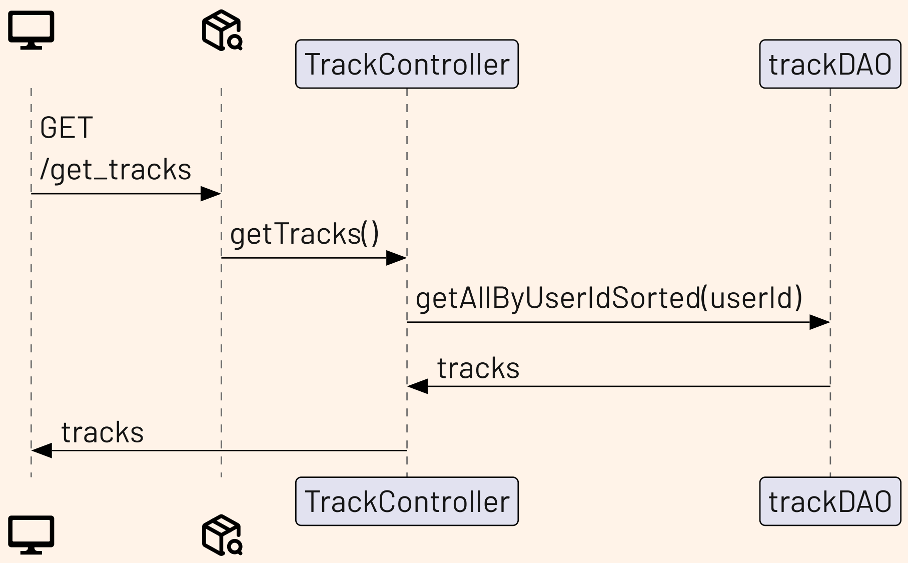
Comment

6.10 GetTrackById sequence diagram



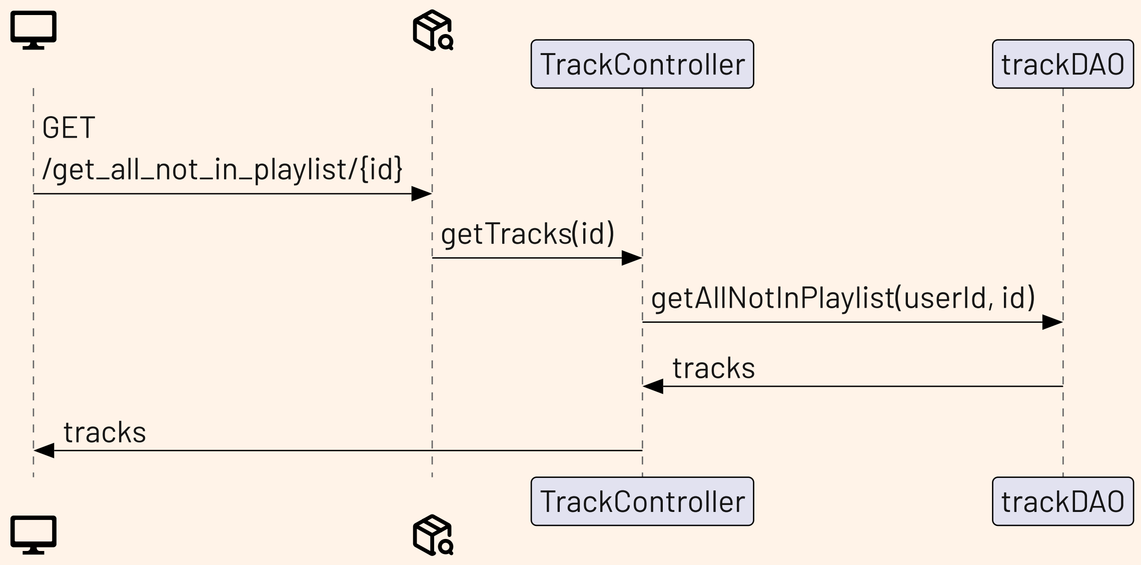
Comment

6.11 GetTracks sequence diagram



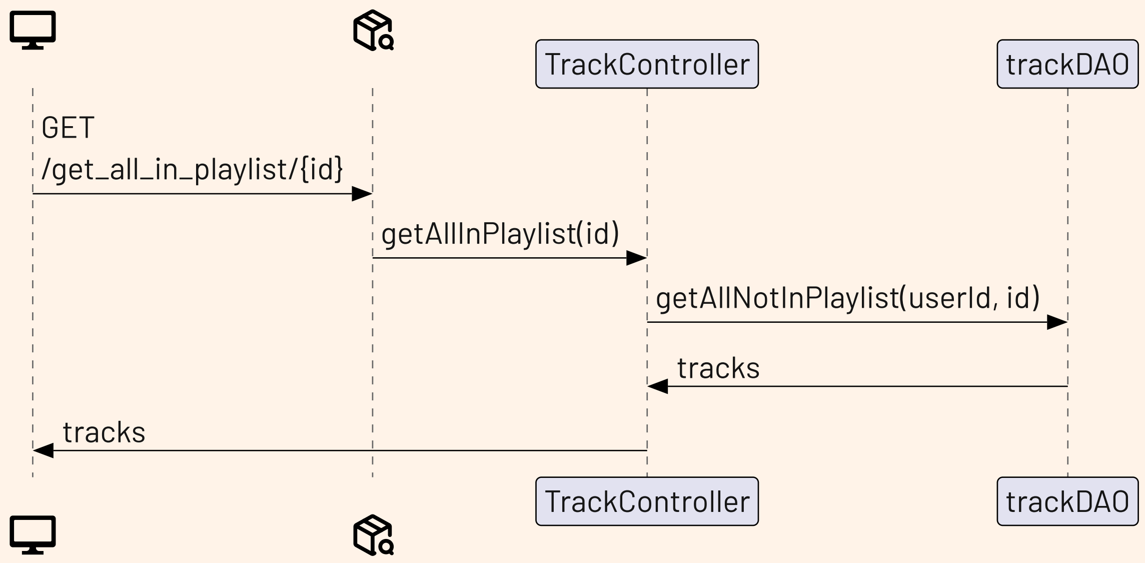
Comment

6.12 GetAllNotInPlaylist sequence diagram



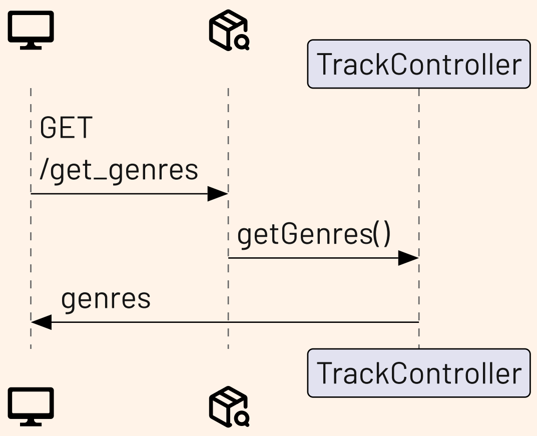
Comment

6.13 GetAllNotInPlaylist sequence diagram



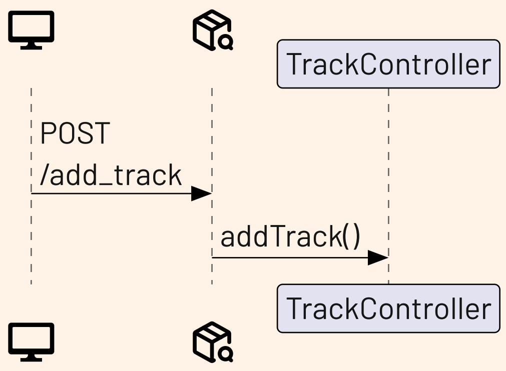
Comment

6.14 GetGenres sequence diagram



Comment

6.15 AddTrack sequence diagram



Comment