Web Technologies Project @ PoliMi, 2025

Creating a Playlist Manager with Thymeleaf & JS

Vlad Raileanu
raileanu.vlad29@gmail.com
https://github.com/rokuban

Vittorio Robecchi
vittorio.robecchi@gmail.com
https://github.com/VictuarVi

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 listem to them.

It should be noted there are two versions: a **only-HTML version**, which is structured as a series of seperate webpages; and a **JS version**, which is structured a single-page webapp. The functionalities are mostly the same, the code changes at a frontend level. For more information see Section 1.

Both of the them feature the same CSS code (see Section 5).

Tools To create the project, our professor decided to adopt the following technologies: **Java**, for the backend server with servlets leveraging Jakarta's API capabilities; **Thymeleaf**, a template engine; and **Apache Tomcat**, to run the server.

Some liberties were taken and we decided to use **MariaDB** for the database[1] instead of MySQL, since the former is a open source fork of MySQL, one of the most widely used DBMS.

Last but absolutely not least, this very document you are reading now has been typeset with none-other than **Typst** [1], the much needed successor to LaTeX. Also, to create sequence diagrams we made use of the `chronos` package [2].

Configuration & Run In order to run this project, the following packages and their respective versions are to be installed:

- Java JDK 24 [3]
- Apache Maven [4]
- Apache Tomcat 10 [5]
- Thymeleaf [6]
- MariaDB [7]

Then Maven will fetch all the corrected dependencies (such as the JDBC driver). We opted to use IntelliJ Idea Ultimate Edition [8] though there are no restrictions – feel free to use whatever editor you want, even Eclipse, if you must[2]. Once you made sure all the dependencies are correctly installed, let Tomcat deploy the server, which will be found at[3]:

http://localhost:8080/[version]_war_exploded

The credentials are stored in plain text in the database (see Section 3.4), while the tracks and images are stored in `target/webapp` (see Section 3.8).

The repository is bundled with some mock data, which can be found in the corresponding folder at the root of the project. They are copyright free songs [9] because we didn't want to get sued 🤭 .

[1] We also could have used SQLite and go for a static webpage.

[2] *I wrote that* only *out of kindess, since I wouldn't recommed it even to my worst enemy. — victuarvi.*

[3] `[version]` is either `pure_html` or `js` depending on what you run.

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 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 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 uò 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'ordinamentodesiderato, 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 aggunto 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 by **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**.

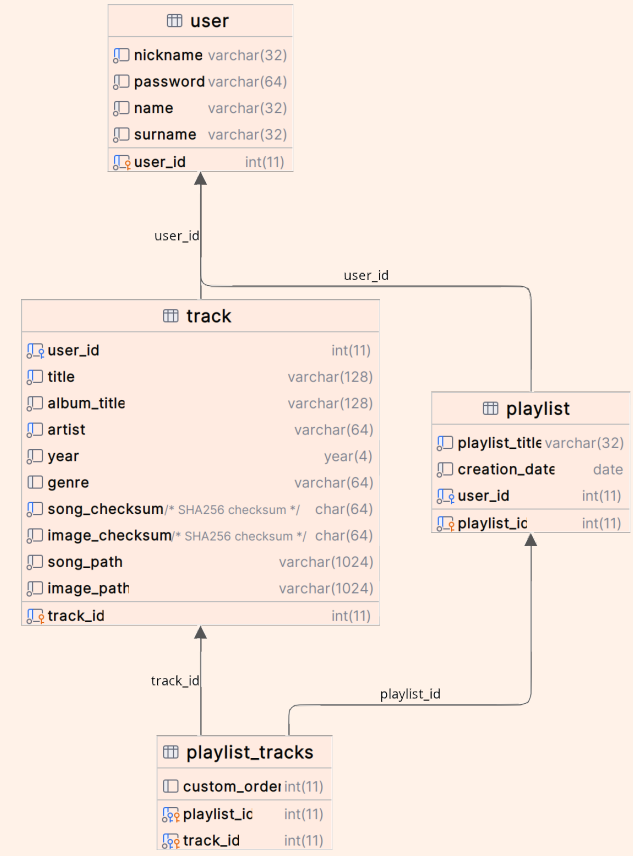


Figure 2: UML diagram.

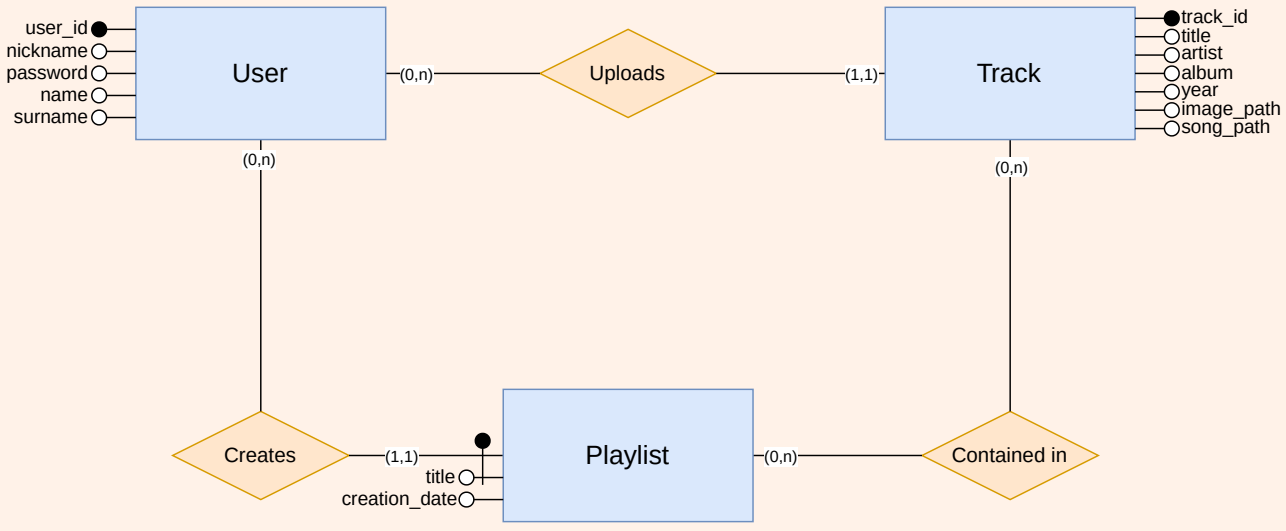


Figure 1: ER diagram, .

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.

- Every cell contains the track's title and album name
- 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.

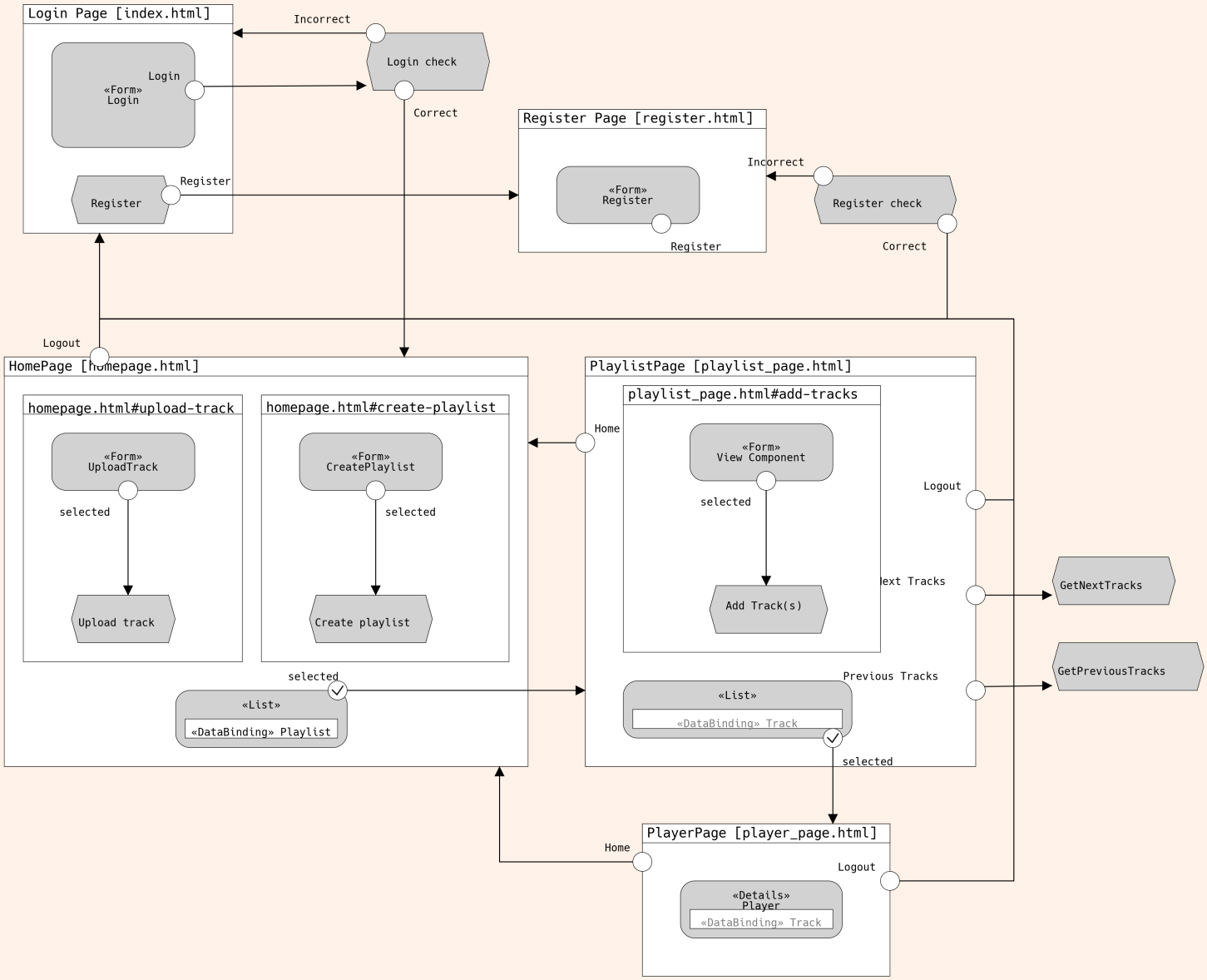


Figure 3: IFML diagram.

3

Sequence diagrams

3.1 Components

The projects is built from the following components:

1. DAOs
    - PlaylistDAO
    - TrackDAO
    - UserDAO
2. Entities
    - Playlist
    - Track
    - User
3. Servlets
    - Login
    - HomePage
    - Playlist
    - Register
    - Track
    - Logout
    - AddTracks
    - CreatePlaylist
4. Filters
    - UserChecker
    - InvalidUserChecker
    - TrackChecker
    - PlaylistChecker

3.2 DAOs methods

PlaylistDAO methods:

- getPlaylistTitle
- deletePlaylist
- getTrackGroup
- addTracksToPlaylist
- removeTracksFromPlaylist
- checkPlaylistOwner
- getUserPlaylists
- getPlaylistTracksByTitle
- createPlaylist
- getPlaylistTracksById

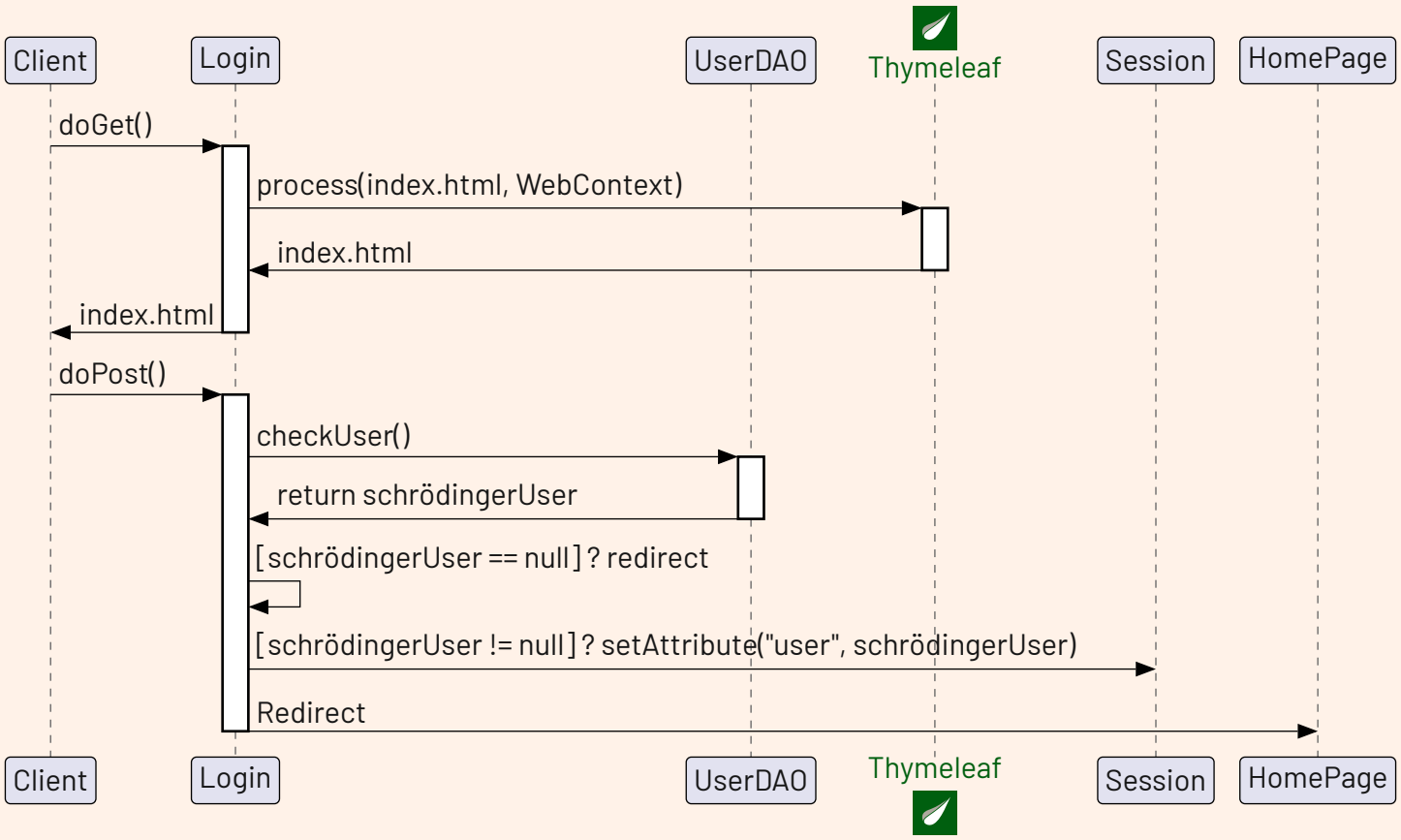
TrackDAO methods:

- addTrack
- isImageFileAlreadyPresent
- checkTrackOwner
- isTrackFileAlreadyPresent
- getTrackById
- getUserTracks

UserDAO methods:

- checkUser
- addUser

3.3 Login sequence diagram

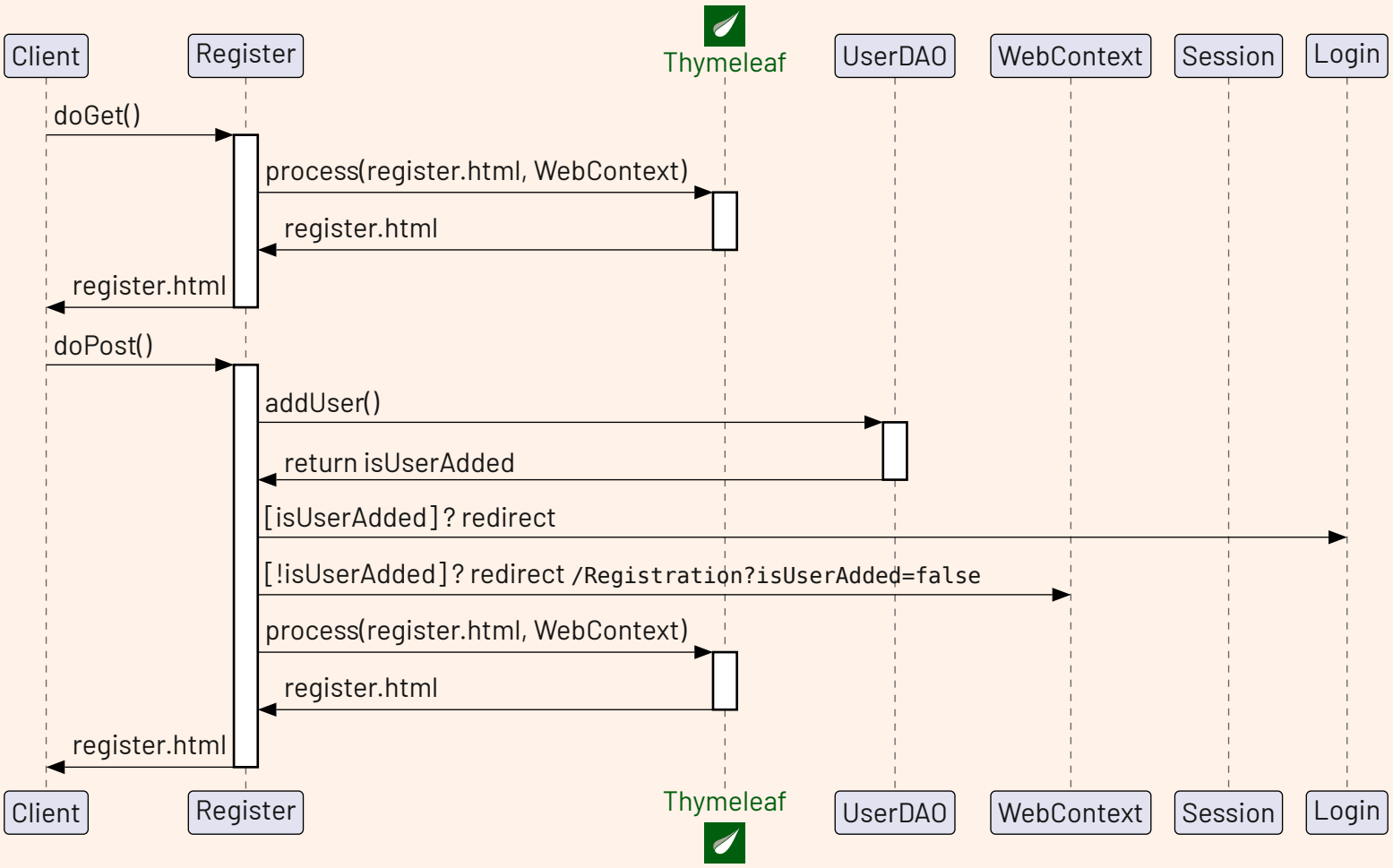


Comment

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 chosen locale.
Afterwards, the User inserts their credentials.

Those values are passed to the `checkUser()` function that returns `schrödingerUser` – as the name implies, the variable might return a User; otherwise `null`. If `null`, then the credentials inserted do not match any record in the database; else the User is redirected to their HomePage and the `user` variable is set for the current session.

3.4 Register sequence diagram



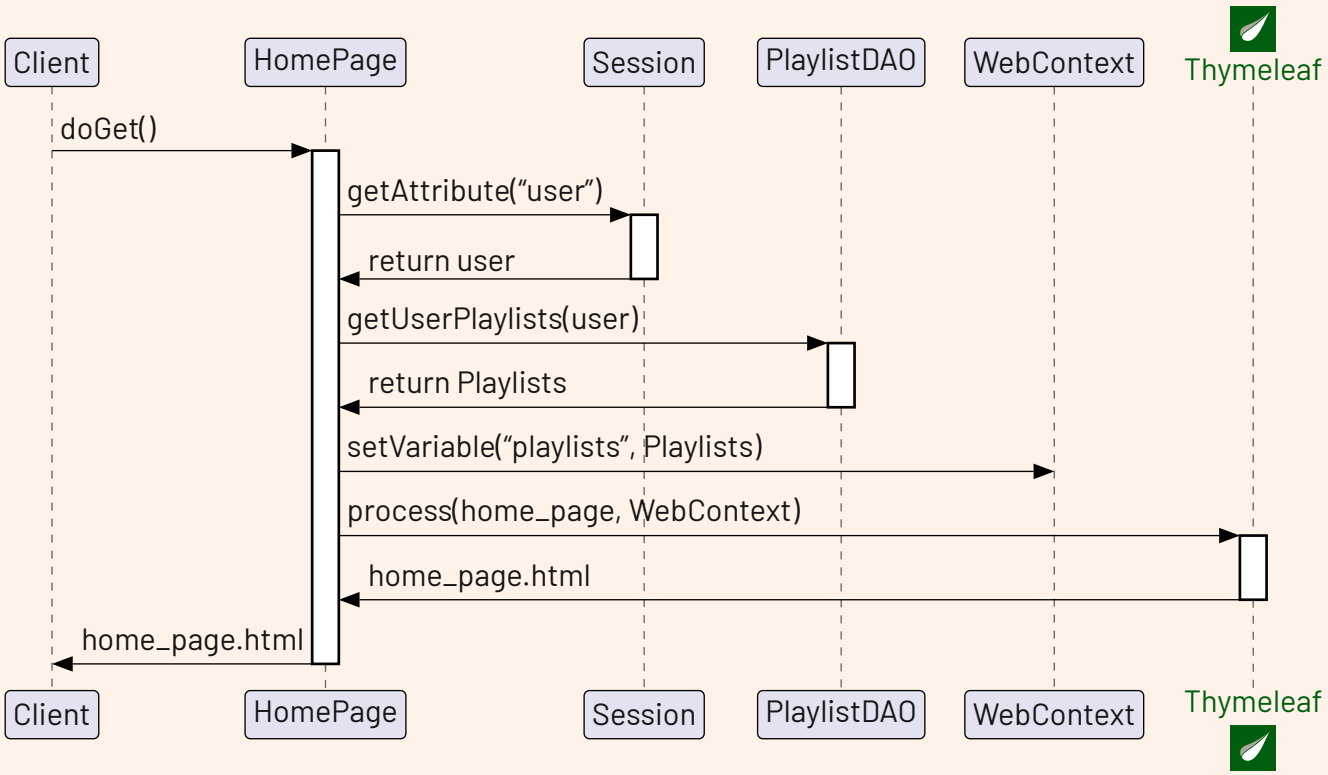
Comment

If the User is not yet registered, they might want to create an account. If that's the case, as per the Login sequence diagram, initially thymeleaf processes the correct context, then the User inserts the credentials.

Depending on the nickname inserted, the operation might fail: there can't be two Users with the same nickname. If that does not happen, then `isUserAdded` is `true`, then there will be the redirection to the Login page.

Else the program appends `isUserAdded` with `false` value and redirects to the Registration servlet: thymeleaf checks for that context variable and if it evaluates to false, it prints an error.

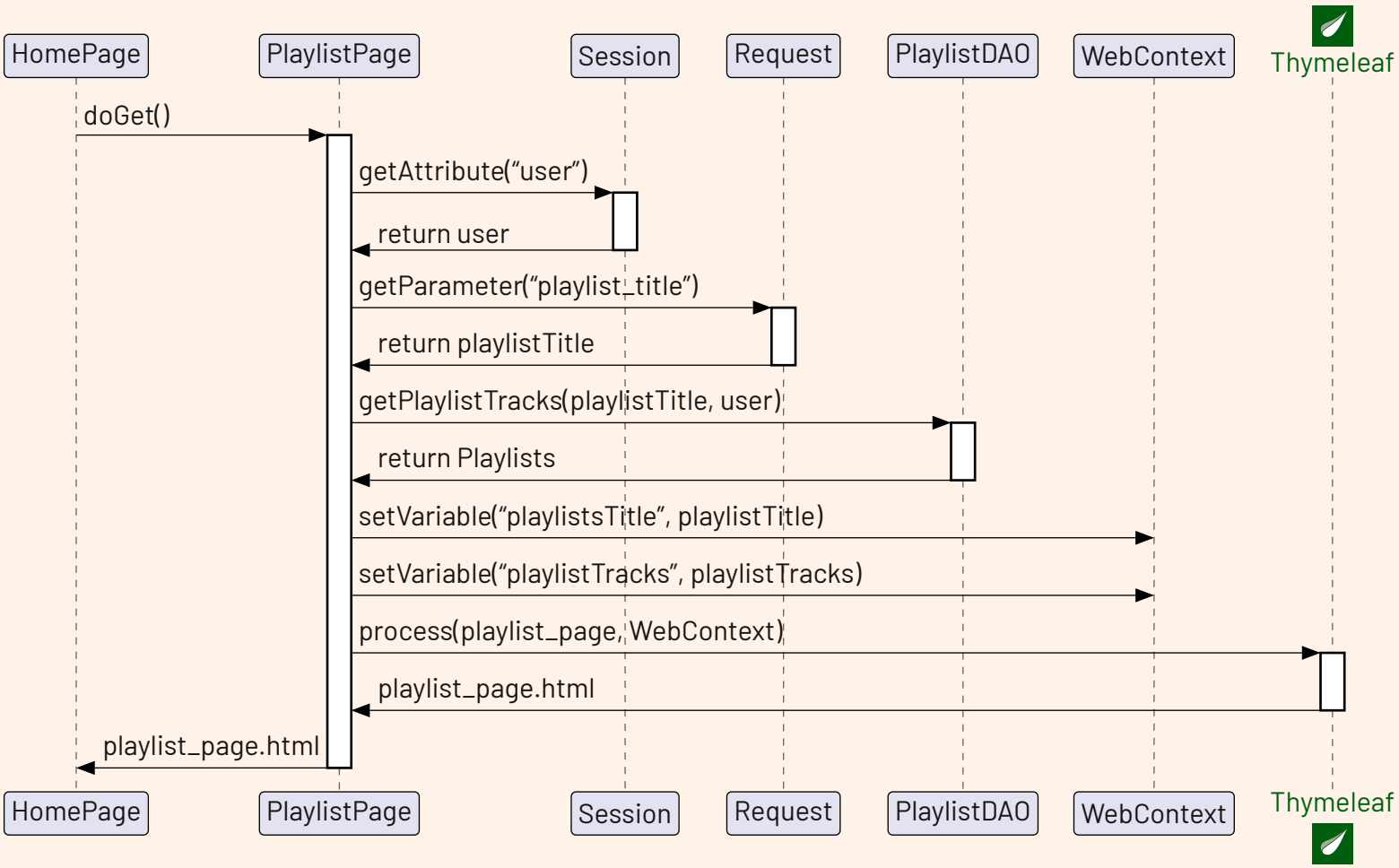
3.5 HomePage sequence diagram



Comment

Once the Login is complete, the User is redirected to their HomePage, which hosts all their Playlists. In order to do so, the program needs to User attribute – which is retrieved via the session; then, it is passed to the `getUserPlaylists` function and finally thymeleaf displays all values.

3.6 PlaylistPage sequence diagram



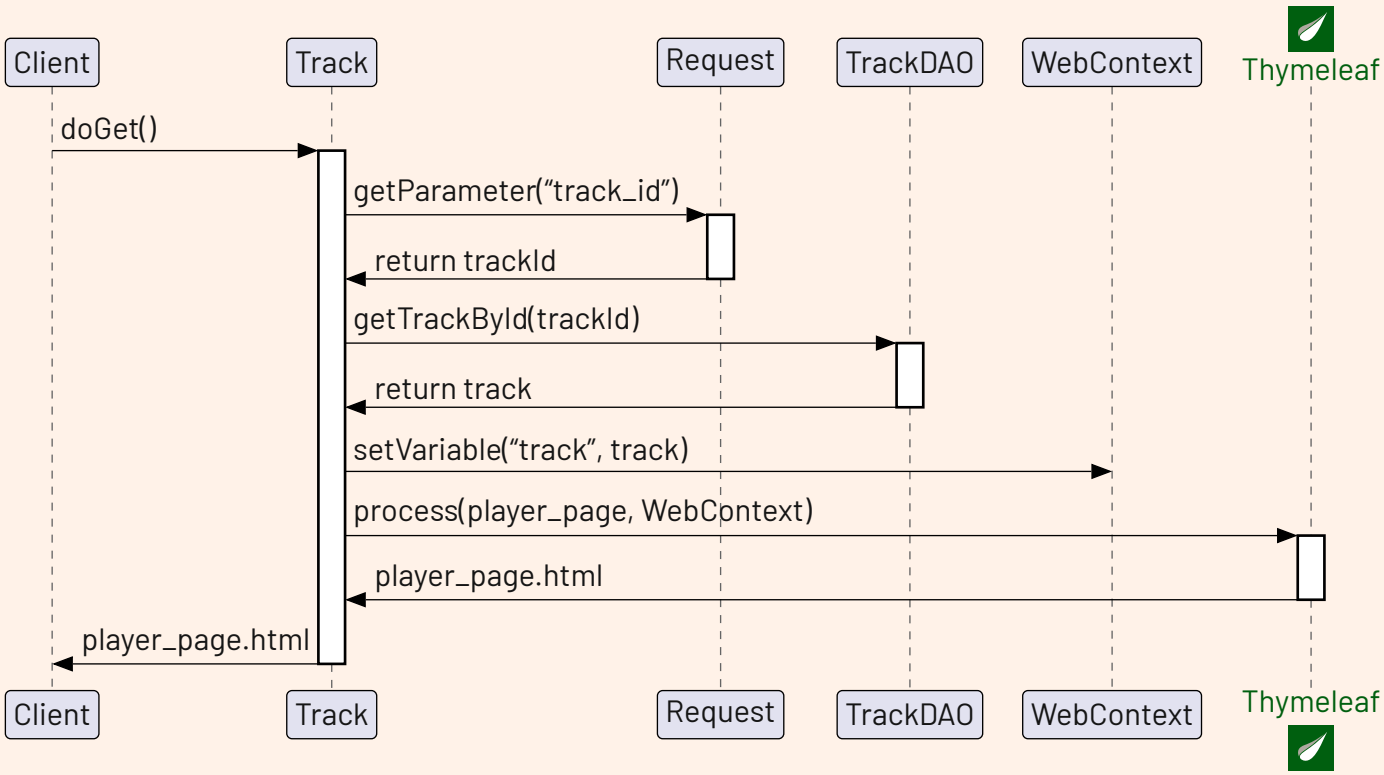
Comment

From the HomePage, the User is able to see all their playlists. By clicking on either one of them, the program redirects to the corresponding PlaylistPage, which lists all the tracks associated to that playlist.

In order to do so, the program needs the User attribute – which is retrieved via the session – and the title of the playlists, which is given as a parameter by pressing the corresponding button in HomePage.

Then those value are passed to `getPlaylistTracks()`, that returns all the tracks. Finally, thymeleaf processes the context and display all the tracks.

3.7 Track sequence diagram

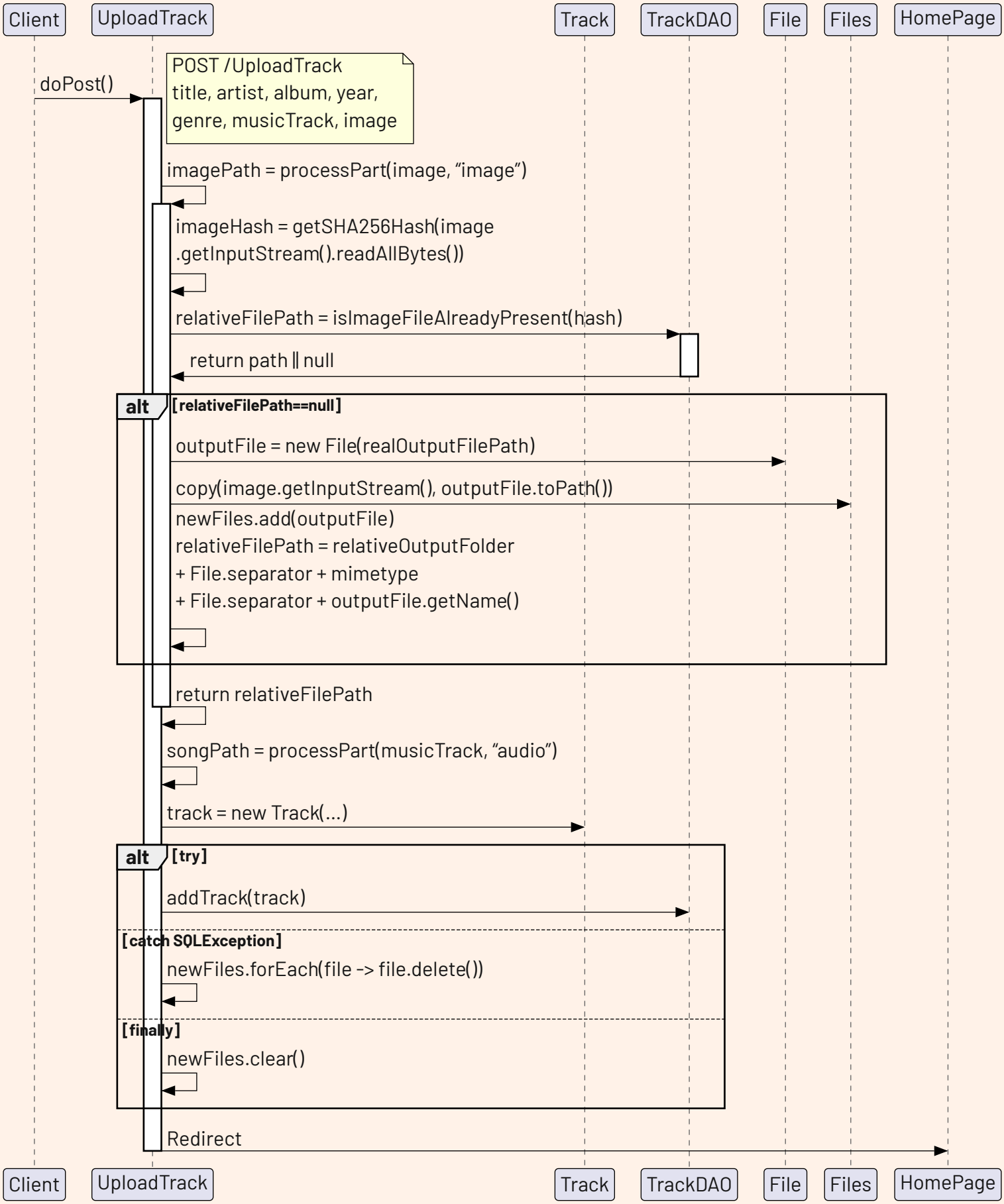


Comment

Once the program has lodead all the tracks associated to a playlist, it allows to play them one by one in the dedicated player page. In a similar fashion to the `getPlaylistTracks()` method, in order to retrieve all the information regarding a single track the program is given the `track_id` parameter by pressing the corresponding button.

Finally, `getTrackById()` returns the track metadata, thymeleaf processes the context and displays all the information.

3.8 UploadTrack sequence diagram



Comment

The User can upload tracks from the appropriate form in the homepage (Section 3.5). When the POST request is received, the request parameters are checked for null values and emptiness (omitted in the diagram for the sake of simplicity), and the uploaded files are written to disk by the `processPart` method, which has two parameters: a Part object, which "represents a part or form item that was received within a multipart/form-data POST request" [10], and its expected MIME type. The latter does not need to be fully specified (i.e. the subtype can be omitted).

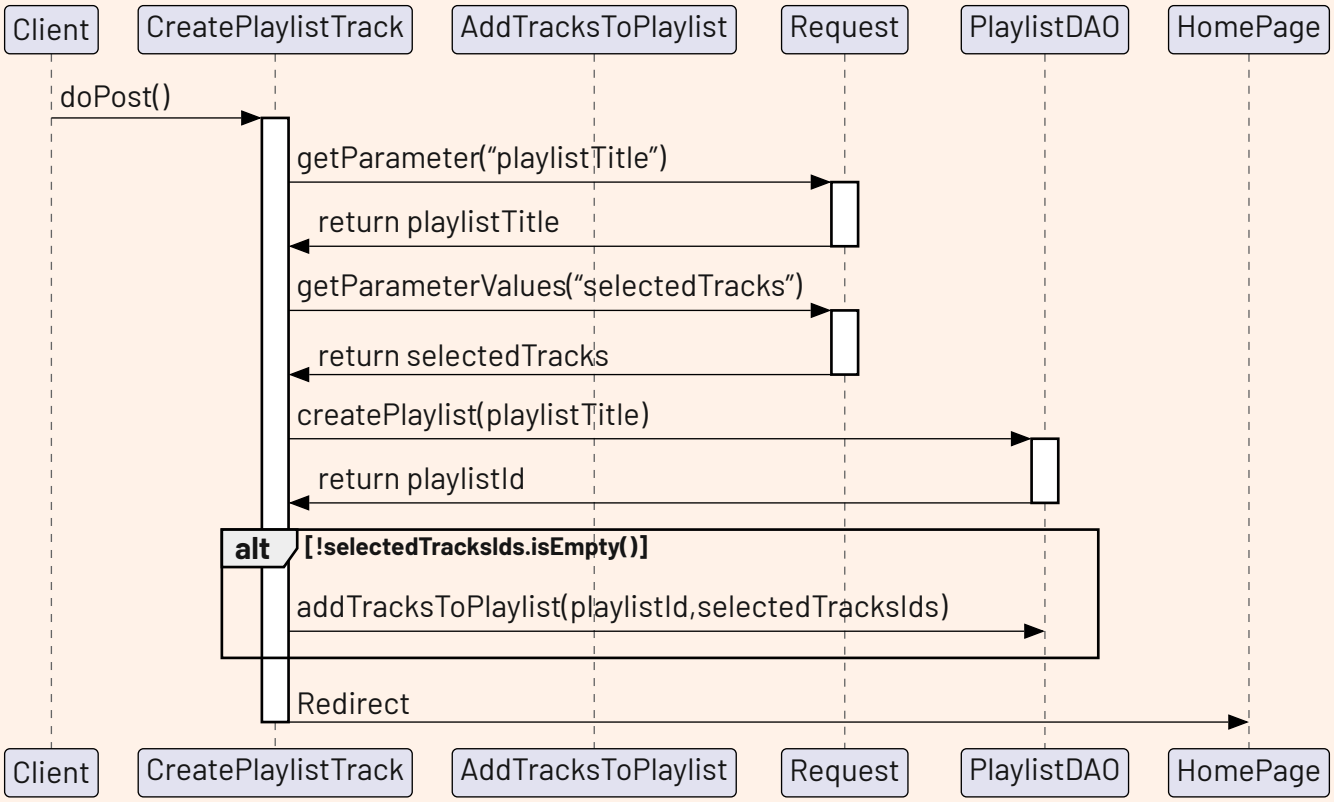
Before writing the file to disk, the method checks for duplicates of the file by calculating its SHA256 hash and querying the database with the two methods: `isTrackFileAlreadyPresent` and `isImageFileAlreadyPresent`; present in TrackDAO.

Those two return the relative file path corresponding to the file hash if a matching one is found, otherwise null. In the former case, `processPart` returns the found path and the new track is uploaded using the already present file, this avoiding creating duplicates; in the latter case `processPart` proceeds by writing the file to disk and returning the new file's path.

*To write the file to the correct path in the webapp folder (`realOutputFolder`), the method `context.getRealPath(relativeOutputFolder)` is called, where `relativeOutputFolder` is obtained from the `web.xml` file and is, in our case, `"uploads"`; `realOutputFolder` is obtained by appending, with the needed separators, the MIME type to the result of `getRealPath`; to get `realOutputFilePath`, a random UUID and the file extension are appended to `realOutputFolder`. Having obtained the desired path, the file can be created and then written with the `Files.copy` method. The file can be found in target/*artifactId-version*/uploads/ in the project folder.*

In conclusion, `processPart` adds the new file to the newFiles list in `UploadTrack` and returns the path relative to the webapp folder because that's where the application will be looking for when it has to retrieve files. Once this is completed, the new Track object is created and passed to the `addTrack` method of TrackDAO; if an `SQLException` is thrown, all the files in `newFiles` list are deleted and then, in the finally block, the list is cleared.

3.9 CreatePlaylist sequence diagram

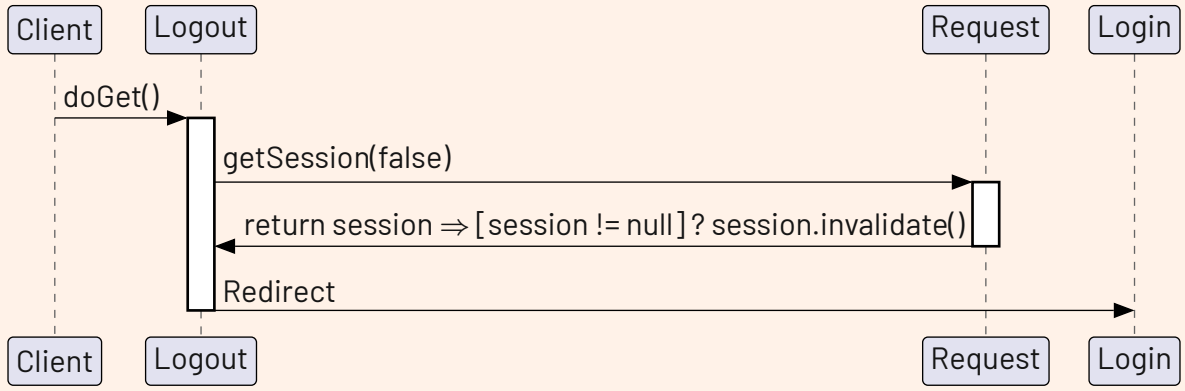


Comment

The user can create playlists with the appropriate form in the homepage. There, a title needs to be inserted and, optionally, one or more tracks can be chosen from the ones uploaded by the user. When the servlet gets the POST request, it interacts with the PlaylistDAO to create the playlist with the createPlaylist method and to add the selected tracks with the addTracksToPlaylist method.

Note that selectedTracksIds is a list of integers obtained by converting the strings inside the array returned by getParameterValues("selectedTracks") with the Integer.parseInt method.

3.10 Logout sequence diagram



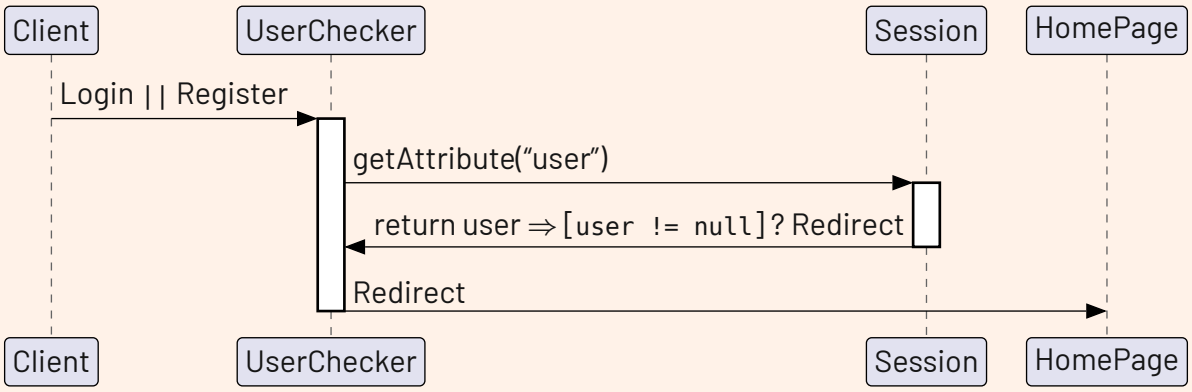
Comment

From every web page except Login and Register, the User is able to logout, at any moment. It's a simple `GET` request to the Logout servlet, which checks if the `user` session attribute exists; if it does, then it invalidates the session and redirects the User to the Login page.

4

Filter mappings

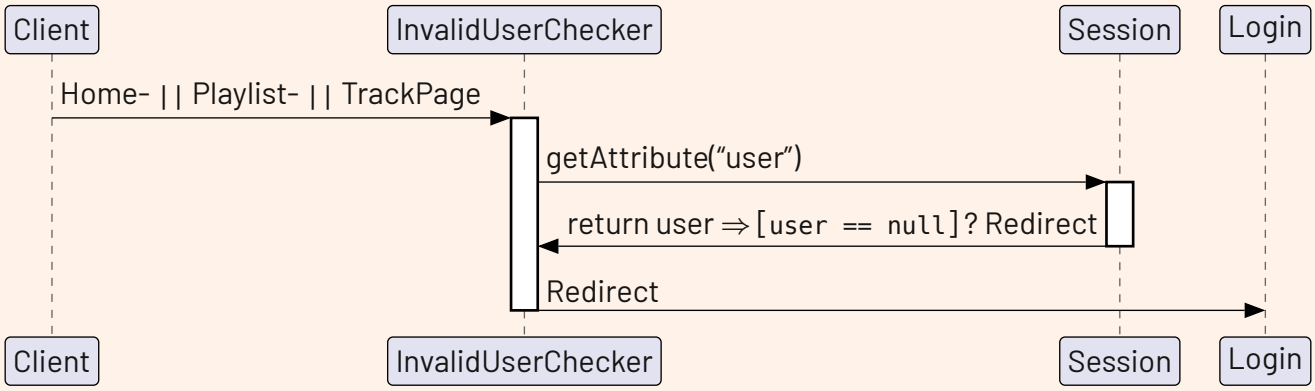
4.1 UserChecker filter



Comment

The `UserChecker` filter checks, once the client accesses the Login or Register webpage, if the User is logged. If that's the case, then the program redirects to the HomePage. If not, then the `InvalidUserChecker` filter comes in.

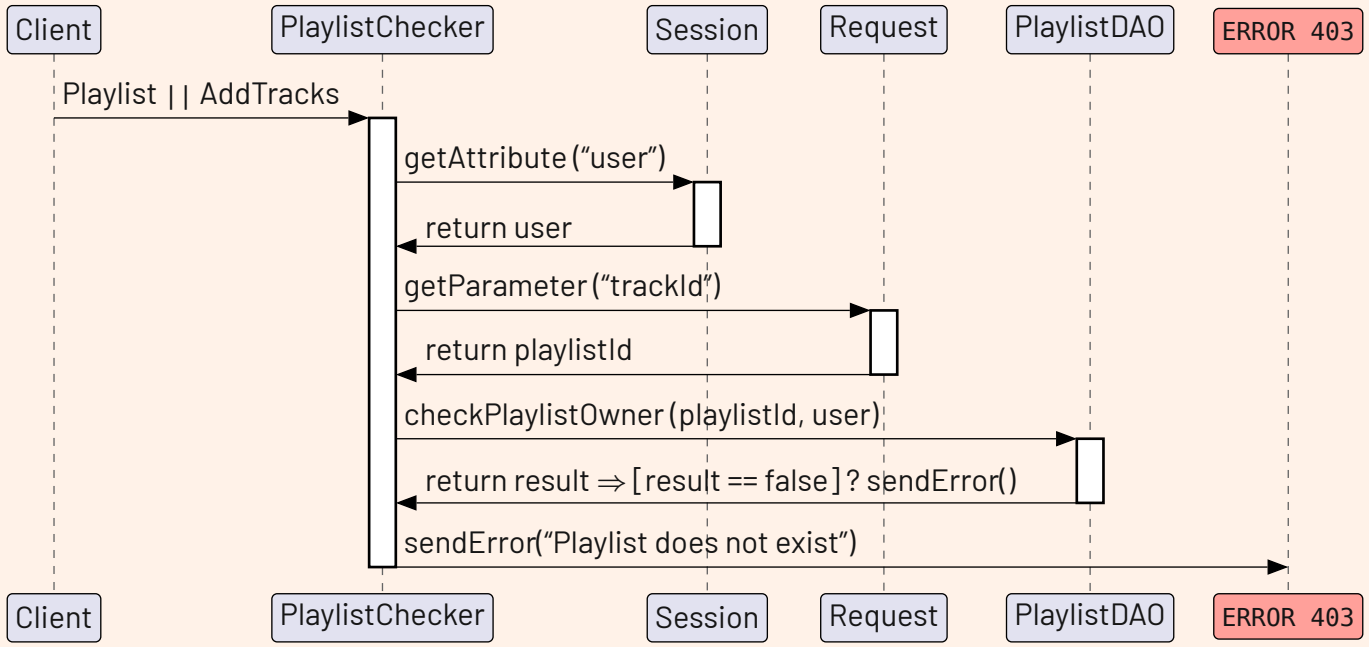
4.2 InvalidUserChecker filter



Comment

The `InvalidUserChecker` filter does the exact opposite of `UserChecker`. If the client accesses pages all the other pages – HomePage, PlaylistPage, TrackPage – and is not logged in, *then the program redirects to the Login page.*

4.3 PlaylistChecker filter



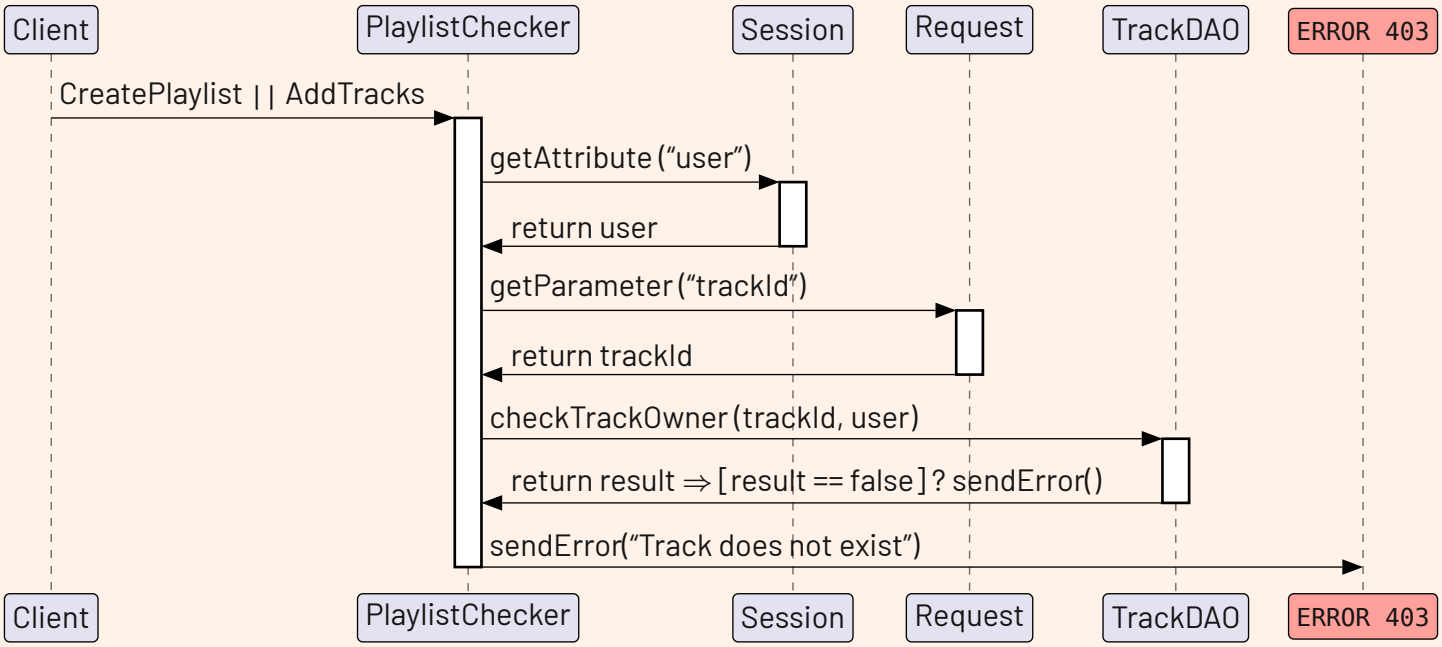
Comment

The `PlaylistChecker` filter is invoked in two scenario: after the User has clicked on a playlist on HomePage (Section 3.6) and when uploading a track (Section 3.8).

It is in charge of checking if the requested playlist actually belongs *to the User requesting or trying to upload it. This is done via obtaining the User attribute from the session – which is impossibile without extending the `HttpServlet` or `HttpFilter` classes – and getting the needed paramaters from the request.*

Finally, a query is performed against the database. If the result is false, then the server will respond with `ERROR 403: forbidden`.

4.4 TrackChecker filter



Comment

Even the `TrackChecker` filter is invoked in two scenario: during the creation of a playlist (Section 3.9) and during the UploadTrack sequence (Section 3.8).

`TrackChecker` applies a very similar pipeline `PlaylistChecker`: instead of checking the playlist, it does the same job but for one of more tracks when the User requests to add them to a playlist or when the User request to play one.

Again similarly to `PlaylistChecker`, tt also obtains the User attribute from the session and the needed parameters; if the User does not have access rights to the requested track(s), the response is `ERROR 403`.

5

Cascading Style Sheets (CSS) styling

5.1 Introduction

The project is based on a single CSS file, `components.css`, and all the others rely upon it to retrieve the styles. Furthermore, all the colours are sourced from the `colors.css` file, which is based on *tinted-theming* [11], a collection of commonly used themes in the developing world. We have chosen to use the *Classic Light theme*[4].

If you want to change the overall theme of the website, just switch to a new colorscheme by looking at the tinted-theming gallery. In `colors.css` there are a few commented styles to choose from.

```
body {
    background-color: var(--default-
background);
    padding: 1rem 2rem 2rem 2rem;
    line-height: 1.6;
    word-spacing: 1px;
    font-family: "JetBrains Mono",
monospace;
    height: 100vh;
    text-overflow: ellipsis;
}
```

As stated earlier, the `background-color` is sourced from the `colors.css`. Then the padding is always `2rem`, except above, where it's `1rem`. The text is able to wrap thanks to `ellipsis` option on `text-overflow`.

After the body, we styled all the elements in a consistent manner.

5.2 Buttons

```
.button {
    color: var(--selection-background);
    background-color: var(--default-
foreground);
    border: 2px solid var(--dark-
foreground);
    height: 3rem;
    border-radius: 6px;
    font-weight: bold;
    vertical-align: middle;
    margin: 0.5rem 0 0.5rem 0;
    padding: 1em;
    font-family: "JetBrains Mono",
monospace;
}
```

Every button is derived from the one above. The text is aligned in the center both horizontally and vertically; its weight set to bold. Then there are some margin and padding to help the user see better[5]*.*

A notable exception to the buttons colorscheme is the `logout` button:

```
.logout {
    background-color: var(--variables);
    font-weight: bolder;
    color: var(--lighter-background);
}

.logout:hover {
    background-color: var(--data-types);
}
```

Both the `background-color`, `font-weight` and `color` are different, to further imply that the logout button is different from the others (upload track, create playlist...).

5.3 Containers

The first container the user sees is the Login one, which shares its design with Register and the track player:

```
.center-panel {
    width: 300px;
    background-color: var(--lighter-
background);
    border: 1px solid var(--dark-
foreground);
    padding: 3rem;
```

[4] This very documentation also is sourced from the exact same colourscheme.

[5] There will be later an exception.

```
    text-align: center;
}
```

An important aspect of login and register is their horizontal bar:

```
hr {
  display: block;
  height: 1px;
  border: 0;
  border-top: 1px solid var(--light-
background);
  margin: 1em 0;
  padding: 0;
}
```

which is not used in the track player.

A basic function of a Playlist Manager is being able to display all the playlists and tracks of a given user. To achieve that, we opted for a classic layout composed of a top and bottom navigation bars and a main, central section.

```
.nav-bar {
    width: 100%;
    margin: 0;
    display: flex;
    flex-wrap: wrap;
    align-content: space-around;
    justify-content: center;
    align-items: center;
    gap: 1rem;
}
```

The navigation bar is the same both above and below. It's a flex container because it's important to have a flexible container for the main-title (e.g. "All Playlists") and the buttons (with a variable number between screens).

The layout is computed as follows:

title	*spacer*	*button*	*button*	*logout*

so we created the `spacer` element:

```
.spacer {
    flex-grow: 1;
}
```

which takes all the space available.

Next, the tracks and playlists containers.

```
.items-container {
    width: 100%;
    display: grid;
    grid-template-columns: 1fr 1fr 1fr
1fr 1fr;
    align-content: baseline;
    justify-content: center;
    gap: 1rem;
    padding: 1rem 0 1rem 0;
}

.single-item {
    display: flex;
    flex-wrap: nowrap;
    background-color: var(--light-
background);
    border: 2px solid var(--data-types);
    border-radius: 5px;
    color: var(--lighter-background);
    padding: 1rem;
    height: 150px;
    font-family: "JetBrains Mono",
monospace;
    font-weight: 700;
    text-align: left;
    align-content: end;
    align-items: end;
    justify-content: space-between;
}

.single-item:hover {
    background-color: var(--variables);
    cursor: pointer;
}
```

According to project the specifications (Section 2), there must be at most 5 tracks per page*: we opted for a CSS grid. This works well along with the `body` previously set because the grid can expand and shrink its items accordingly.*

As per the navigation bar, the single items are themselves flexible boxes. The difference lies in the fact they are not allowed to wrap – one might ask: why not, since the tracks must list both track title and album title? because we handle that line break manually with the `<br>` tag.

5.4 Modal

Finally, without a doubt the most difficult CSS component in this project is the modal, which is a dialog window created entirely with CSS. As a complex element, it can be broken in multiple parts:

- The window

```
.modal-window {
    position: fixed;
    background-color: rgba(255, 255, 255,
0.25);
    top: 0;
    right: 0;
    bottom: 0;
    left: 0;
    z-index: 999;
    visibility: hidden;
    pointer-events: none;
    transition: all 0.5s;
}
```

it's `hidden` by default, but once it's invoked it must be be above everything – this is handled by the `z-index` property. Its position must be `fixed`, since it's not a movable window; also it can't be targeted by cursor: `pointer-events` are none. Another key aspect is the background color: in order to make it stand from its background, a slight blurred white is needed:

- The target, when the user presses a button that launches the modal (e.g. Upload Track)

```
.modal-window:target {
    visibility: visible;
    opacity: 1;
    pointer-events: auto;
}

.modal-window > div {
    width: 400px;
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    padding: 1em;
    background: var(--lighter-background);
    border: 2px solid var(--variables);
}
```

once the modal has been invoked, its visibility must be switched to `visible` and `opacity` to 1. The child element `div` of the window must at the center of screen, both horizontally and vertically: this is managed with the `top`, `left` and `translate` properties.

- The close button

```
.modal-close {
    color: var(--lighter-background);
    background-color: var(--variables);
    border-radius: 5px;
    position: absolute;
    top: 2%;
    right: 2%;
    cursor: pointer;
    padding: 0.2rem;
    font-size: 0.8rem;
    font-weight: bold;
    text-align: center;
    text-decoration: none;
}

.modal-close:hover {
    color: black;
}
```

as stated previously, the `modal-close` button is an exception to the `button` rule. It's considerably smaller than the others, the cursor is immediately `pointer`. Its position is computed on the `modal-window`, from above right.

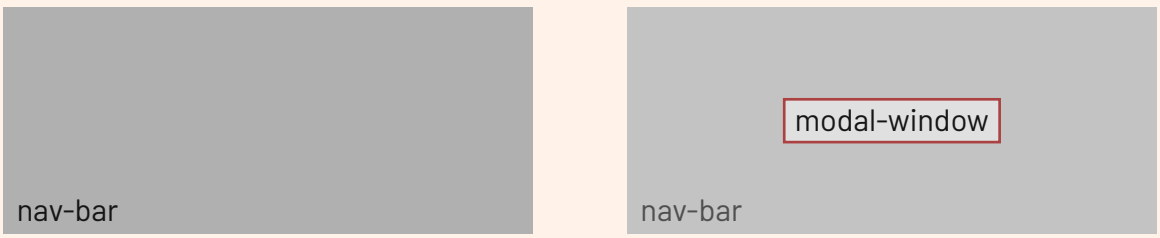


Figure 16: Modal representation.

Bibliography

[1] L. Mädje, M. Haug, and The Typst Project Developers, *Typst*. [Online]. Available: https://github.com/typst/typst

[2] Louis Heredero, *chronos (typst package)*. [Online]. Available: https://typst.app/universe/package/chronos

[3] "Java Development Kit." [Online]. Available: https://openjdk.java.net/

[4] "Apache Maven." [Online]. Available: https://maven.apache.org/

[5] "Apache Tomcat." [Online]. Available: https://tomcat.apache.org/

[6] "Thymeleaf." [Online]. Available: https://www.thymeleaf.org/

[7] "MariaDB." [Online]. Available: https://mariadb.org/

[8] Vittorio Robecchi, *Web Technologies IntelliJ Guide @ PoliMi*. [Online]. Available: https://github.com/VictuarVi/wt-intellij-guide

[9] *NoCopyrightSounds*. [Online]. Available: https://ncs.io/

[10] "Part interface." [Online]. Available: https://jakarta.ee/specifications/servlet/6.1/apidocs/jakarta.servlet/jakarta/servlet/http/part

[11] *Tinted Theming*. [Online]. Available: https://github.com/tinted-theming/home