# TP - Next - App - Router - part 2

Here we want to add some functionalities to our ssr-movie-app

Indeed we want the user to be able to build a playlist of movies and save them in a database so so when they could share the playlist link like `http:localhost:3000/playlists/[playlistId]`.

When a user will ask the playlist page we'll render a table with the movies

## Add `playlists/creation` link to our navigation header to our website

We will add a playlist page, so let's add its link in the header nav bar :

```
<Link href="/playlists/creation" prefetch={false}>
  Playlist creation
</Link>
```

## Create that playlists/creation page

Let's return `Playlist works` for now.

Don't forget the metadata to set the page title and description.

## Provide a way to the user for adding movies

We want the user to be able to store a playlist in database.

But first we need to build it, so we need to handle a list of movies somehow.

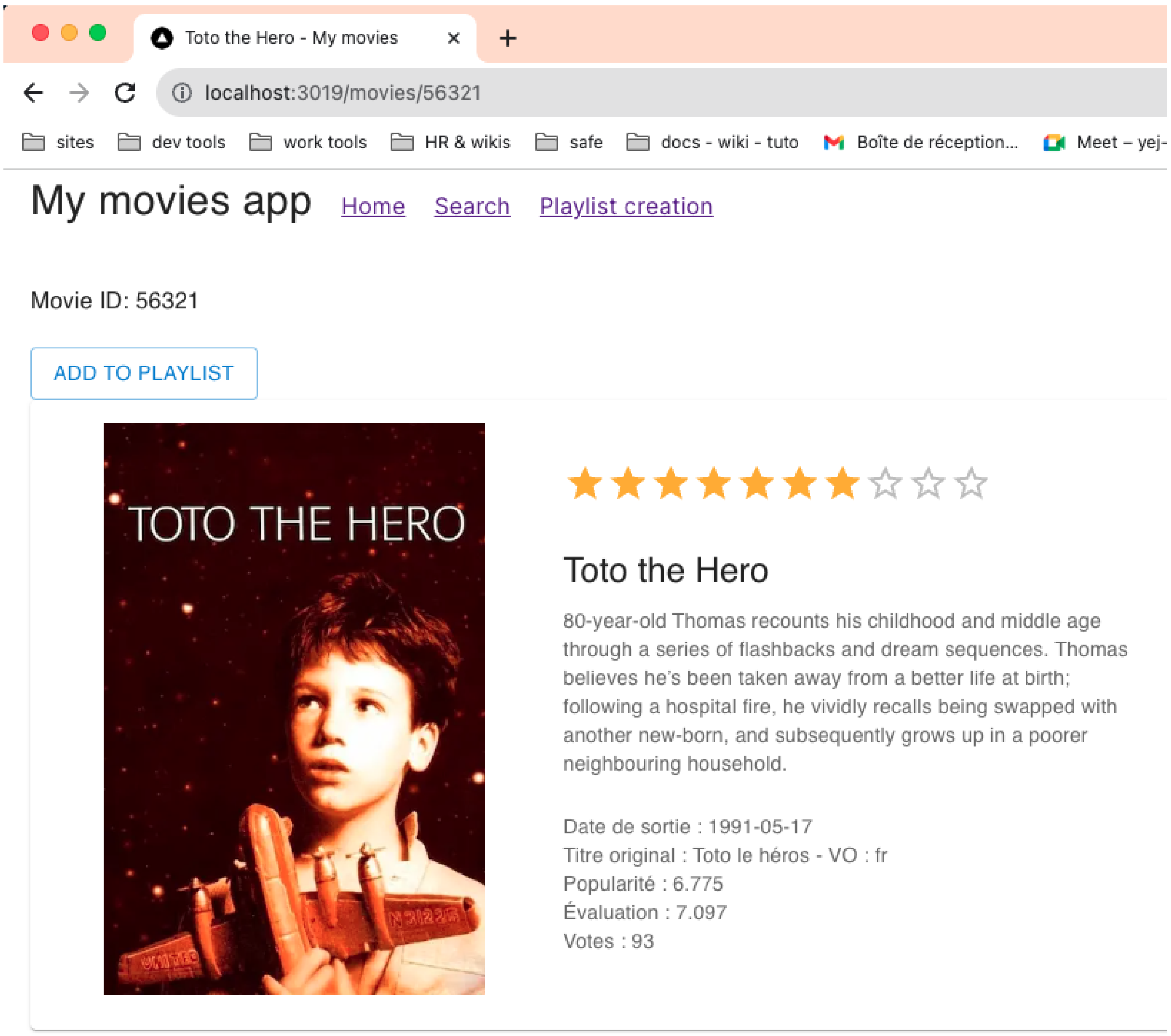
How ??

State, context, local storage, database ? Think about pros and cons of these storage strategies before continuing, and don't forget to discuss about these with me.

Then add a button `Add to playlist` on the movie details page, where the onClick will use our strategy to handle the movie list before storing any playlist. But for now simply log the movie in browser console.

This button has to perform client side because it handles a DOM event: the user click, so it has to be wrapped in a client component, let's call this component `AddToPlaylistButton`

At this point you should have something like this:



Movie ID: 56321

Toto the Hero

80-year-old Thomas recounts his childhood and middle age through a series of flashbacks and dream sequences. Thomas believes he's been taken away from a better life at birth; following a hospital fire, he vividly recalls being swapped with another new-born, and subsequently grows up in a poorer neighbouring household.

Date de sortie : 1991-05-17
Titre original : Toto le héros - VO : fr
Popularité : 6.775
Évaluation : 7.097
Votes : 93

## Storage strategy

We will use the local storage, why ? Ask me if you didn't already discussed about it.

Now you can implement the onClick handler of `AddToPlaylistButton`

The local storage of the **browser** is a table as `key` : `value` where the values are string so you need to `stringify` when you set a value and `parse` when you get the value. Also, the local storage don't exist on server side. That said, no worries since the button is a client component, it can deal with the browser local storage.

To deal with the local storage, it's very simple:

- Store a value:

`window.localStorage.setItem(THE_ITEM_KEY, THE_VALUE)`

Don't forget, local storage handle only string so if you want to store an `array` or an `object` , you first need to stringify them like this

const myArrayAsString = `JSON.stringify(MY_ARRAY)` , same in case of object

- Retrieve a value

const myValueAsString = `window.localStorage.getItem([The_Item_Key])`

Don't forget then to parse the value if you expect it to be an `array` or an `object`

const myObject (or myArray) = `JSON.parse(myValueAsString)`

**A user shouldn't be able to add a movie twice, each movie should be unique in a playlist !**

Let's create a dedicated service for dealing with local storage:

- `services/localStorage/index.ts` handling whole the local storage, basically it will just export the `localStoragePlaylist` from file below
- `services/localStorage/playlist.ts` dedicated to `playlist` key.
- The model of the playlist in local storage `models/LocalStoragePlaylist.ts`

```
import Movie from "./Movie";

type LocalStoragePlaylist = {
  name: string;
  description: string;
  movies: Movie[];
};

export default LocalStoragePlaylist;
```

### Important

Remember, with Next the page is rendered server side.

But the local storage exists only frontend side, so in your backend execution you won’t be able to access browser’s tool like `document` or `window`

▼ Solution for `LocalStorageService`

- `services/localStorage/index.ts` handling whole the local storage

```
import localStoragePlaylist from "./playlist";

const LocalStorageService = {
  playlist: localStoragePlaylist,
};

export default LocalStorageService;
```

- `services/localStorage/playlist.ts` dedicated to `playlist` key

```
import LocalStoragePlaylist from "@/models/LocalStoragePlaylist";
import Movie from "@/models/Movie";

const localStorageKey = "playlist";

const defaultPlaylist: LocalStoragePlaylist = {
  name: "",
  description: "",
  movies: [],
};

function get(): LocalStoragePlaylist {
  const stringifiedPlaylist = window.localStorage.getItem(localStorageKey);
  if (!stringifiedPlaylist) {
    return defaultPlaylist;
  }

  const playlist = JSON.parse(stringifiedPlaylist) as LocalStoragePlaylist;
  return playlist;
}
```

```
function set(playlist: LocalStoragePlaylist) {
  window.localStorage.setItem(localStorageKey, JSON.stringify(playlist));
}

function addMovie(movie: Movie): LocalStoragePlaylist {
  const currentPlaylist = get();
  const updatedPlaylist = {
    ...currentPlaylist,
    movies: [...currentPlaylist.movies.filter((m) => m.id !== movie.id), movie],
  };
  set(updatedPlaylist);

  return updatedPlaylist;
}

const localStoragePlaylist = {
  addMovie,
};

export default localStoragePlaylist;
```

▼ Solution for `AddToPlaylistButton`

```
"use client";

import Movie from "@/models/Movie";
import LocalStorageService from "@/services/localStorage";
import { Button } from "@mui/material";

type Props = {
  movie: Movie;
};

const AddToPlaylistButton = ({ movie }: Props) => {
  const addToPlaylist = () => {
    LocalStorageService.playlist.addMovie(movie.id);
  };

  return (
    <Button variant="outlined" onClick={addToPlaylist}>
      Add to playlist
    </Button>
  );
};

export default AddToPlaylistButton;
```

## Implement the `playlists/creation` page

Remember, with Next the page is rendered server side.

But the local storage is only frontend side.

So we cannot base our page computation on backend on any frontend tools like local storage

That means the page will handle it client side (CSR), via a client component, keeping data from local storage is like fetching some data, just remember that these data are on client side.

What about using…

…z

…zz

…zzz

a useEffect ??

I guess you know what to do at this point ! 🙂

Try to console.log(THE_MOVIES_OF_THE_PLAYLIST)

▼ Solution for playlist creation page

- `app/playlists/creation/page.tsx`

```
import PlaylistSection from "@/components/PlaylistSection";

export const metadata = {
  title: "Playlist creation - My movies",
  description: "Playlist Creation Page",
};

const PlaylistCreationPage = () => {
  return (
    <div>
      <PlaylistSection />
    </div>
  );
};
```

```
export default PlaylistCreationPage;
```

- `components/PlaylistSection/index.tsx`

```
"use client";

import LocalStorageService from "@/services/localStorage";
import { useEffect } from "react";

const PlaylistSection = () => {
  useEffect(() => {
    const playlist = LocalStorageService.playlist.get();
    console.log(playlist);
  }, []);

  return <section>Playlist Section</section>;
};

export default PlaylistSection;
```

Then create a component `PlaylistMovies` which will be in charge of displaying the list of movies almost like the `SearchResult` component but with some variation

It must contain a delete movie button to remove the movie from the playlist. So on click on it, remove the movie from the playlist stored in local storage

▼ Solution

- `components/PlaylistMovies/index.tsx`

```
import Movie from "@/models/Movie";
import {
  TableContainer,
  Paper,
  Table,
  TableHead,
  TableRow,
  TableCell,
  TableBody,
  Button,
} from "@mui/material";
import ClientTableRow from "../ClientTableRow";
import classes from "./index.module.css";

type Props = {
  movies: Movie[];
  onMovieDeletion: (movie: Movie) => void;
};

const PlaylistMovies = ({ movies, onMovieDeletion }: Props) => {
  const handleClickOnDeleteMovie =
    (movie: Movie) => (event: React.MouseEvent<HTMLButtonElement>) => {
      event.stopPropagation();
      onMovieDeletion(movie);
    };

  return (
    <TableContainer className={classes.root} component={Paper}>
      <Table sx={{ minWidth: 650 }} aria-label="movies playlist table">
        <TableHead>
          <TableRow>
            <TableCell align="center" className={classes.tableCell}>
              ID
            </TableCell>
            <TableCell align="center" className={classes.tableCell}>
              Title
            </TableCell>
            <TableCell align="center" className={classes.tableCell}>
              Vote average
            </TableCell>
            <TableCell align="center" className={classes.tableCell}>
              Vote count
            </TableCell>
            <TableCell align="center" className={classes.tableCell}>
              Popularity
            </TableCell>
            <TableCell align="center" className={classes.tableCell}>
              Release date
            </TableCell>
            <TableCell align="center" className={classes.tableCell}>
              Playlist deletion
            </TableCell>
          </TableRow>
        </TableHead>
        <TableBody>
          {movies.map((movie) => (
            <ClientTableRow
              key={movie.id}
              sx={{ "&:last-child td, &:last-child th": { border: 0 } }}
              className={classes.row}
              linkHref={`/movies/${movie.id}`}
            >
              <TableCell align="center">{movie.id}</TableCell>
              <TableCell align="center">{movie.title}</TableCell>
              <TableCell align="center">{movie.vote_average}</TableCell>
              <TableCell align="center">{movie.vote_count}</TableCell>
              <TableCell align="center">{movie.popularity}</TableCell>
              <TableCell align="center">{movie.release_date}</TableCell>
              <TableCell align="center">
                <Button onClick={handleClickOnDeleteMovie(movie)} color="error">
                  Delete
                </Button>
              </TableCell>
            </ClientTableRow>
          ))}
        </TableBody>
      </Table>
    </TableContainer>
  );
};

export default PlaylistMovies;
```

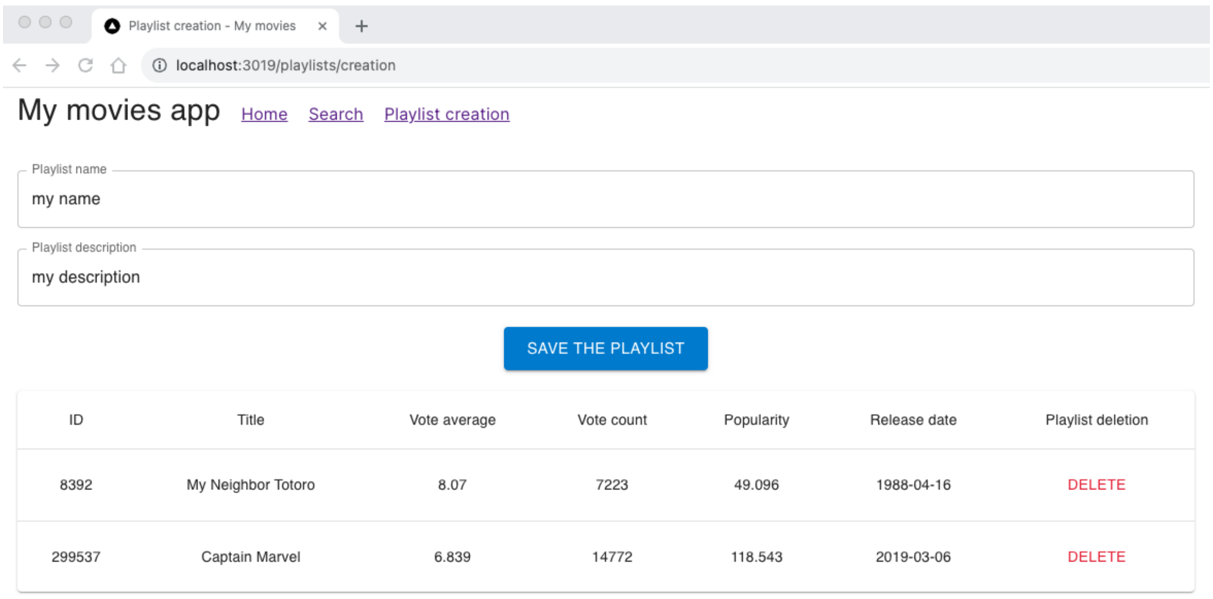
- `components/PlaylistMovies/index.module.css`

```
.root {
  margin: 20px 0;
}

.headCell {
  font-weight: bold;
}

.row {
  cursor: pointer;
}

.row:hover {
  background: #fafafa;
}
```

Once it's done:

- Add an input to keep the playlist name from the user
- Add an input for a description
- Add a button on this page: `Save the playlist` which will console.log the playlist name, description and the movies on click

At this point, you should have something like that

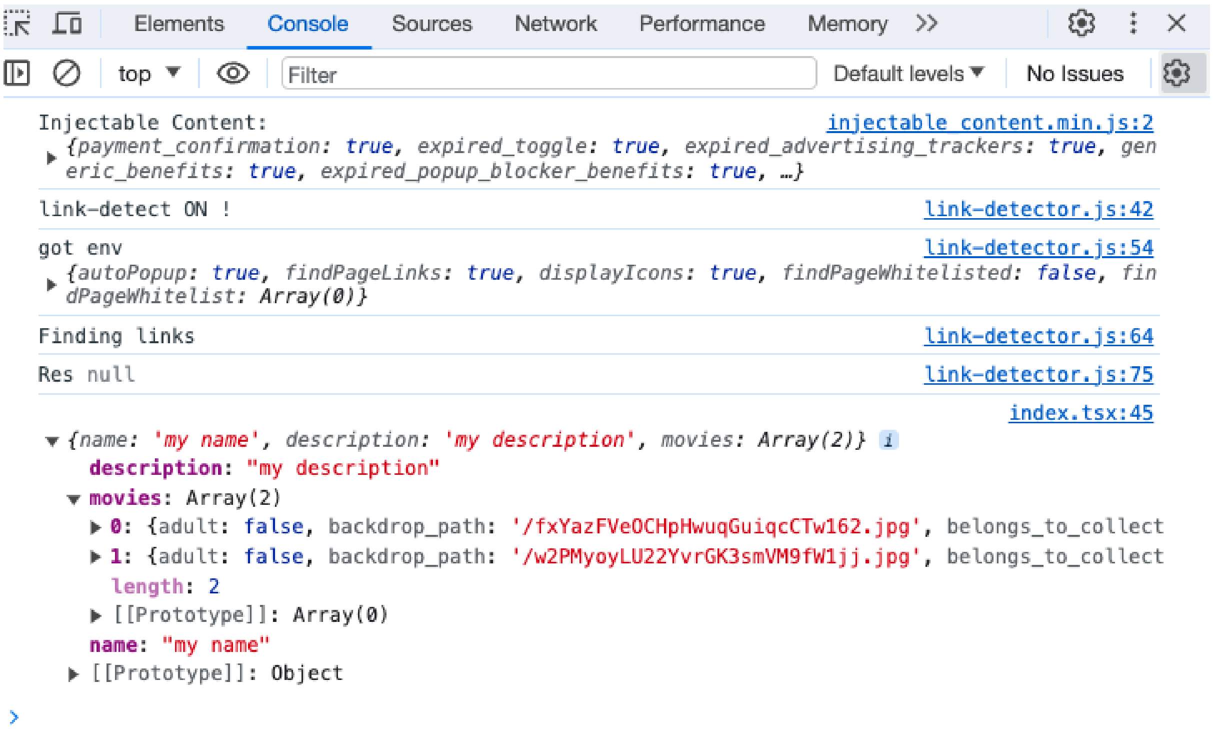

Playlist creation - My movies
localhost:3019/playlists/creation
My movies app
Home
Search
Playlist creation

Playlist name: my name

Playlist description: my description

SAVE THE PLAYLIST

| ID | Title | Vote average | Vote count | Popularity | Release date | Playlist deletion |
|---|---|---|---|---|---|---|
| 8392 | My Neighbor Totoro | 8.07 | 7223 | 49.096 | 1988-04-16 | DELETE |
| 299537 | Captain Marvel | 6.839 | 14772 | 118.543 | 2019-03-06 | DELETE |



```
Injectable Content:                                              injectable_content.min.js:2
▶ {payment_confirmation: true, expired_toggle: true, expired_advertising_trackers: true, gen
  eric_benefits: true, expired_popup_blocker_benefits: true, …}
link-detect ON !                                                         link-detector.js:42
got env                                                                  link-detector.js:54
▶ {autoPopup: true, findPageLinks: true, displayIcons: true, findPageWhitelisted: false, fin
  dPageWhitelist: Array(0)}
Finding links                                                            link-detector.js:64
Res null                                                                 link-detector.js:75
                                                                               index.tsx:45
▼ {name: 'my name', description: 'my description', movies: Array(2)} i
    description: "my description"
  ▼ movies: Array(2)
    ▶ 0: {adult: false, backdrop_path: '/fxYazFVeOCHpHwuqGuiqcCTw162.jpg', belongs_to_collect
    ▶ 1: {adult: false, backdrop_path: '/w2PMyoyLU22YvrGK3smVM9fW1jj.jpg', belongs_to_collect
      length: 2
    ▶ [[Prototype]]: Array(0)
    name: "my name"
  ▶ [[Prototype]]: Object
>
```

▼ Solution for `PlaylistSection` and `LocalStorageService`

- `components/PlaylistSection/index.tsx`

```
"use client";

import LocalStorageService from "@/services/localStorage";
import { useEffect, useRef, useState } from "react";
import PlaylistMovies from "../PlaylistMovies";
import LocalStoragePlaylist from "@/models/LocalStoragePlaylist";
import Movie from "@/models/Movie";
import { Box, Button, TextField } from "@mui/material";

const PlaylistSection = () => {
  const [playlist, setPlaylist] = useState<LocalStoragePlaylist | null>(null);
  const nameInputRef = useRef<HTMLInputElement>(null);
  const descriptionInputRef = useRef<HTMLInputElement>(null);

  useEffect(() => {
    const playlist = LocalStorageService.playlist.get();
    setPlaylist(playlist);
  }, []);

  const handleMovieDeletion = (movie: Movie) => {
    const updatedPlaylist = LocalStorageService.playlist.removeMovie(movie);
    setPlaylist(updatedPlaylist);
  };

  const handleSubmit = (event: React.FormEvent<HTMLFormElement>) => {
    event.preventDefault();

    const name = nameInputRef.current?.value ?? "";
    const description = descriptionInputRef.current?.value ?? "";

    const updatedPlaylist = LocalStorageService.playlist.setMetadata({
      name,
      description,
    });

    setPlaylist(updatedPlaylist);
    console.log(updatedPlaylist);
  };

  return (
    <section>
      {playlist && (
        <>
          <Box
            component="form"
            onSubmit={handleSubmit}
            display="flex"
            flexDirection="column"
            gap="20px"
            alignItems="center"
          >
            <TextField
              inputRef={nameInputRef}
              label="Playlist name"
              fullWidth
              defaultValue={playlist.name}
            />
            <TextField
              inputRef={descriptionInputRef}
              label="Playlist description"
              fullWidth
              defaultValue={playlist.description}
            />
            <Button
              variant="contained"
              type="submit"
              color="primary"
              size="large"
            >
              Save the playlist
            </Button>
          </Box>

          <PlaylistMovies
            movies={playlist.movies}
            onMovieDeletion={handleMovieDeletion}
          />
        </>
      )}
    </section>
  );
};

export default PlaylistSection;
```

- `services/localStorage/playlist.ts`

```
"use client";

import LocalStorageService from "@/services/localStorage";
import { useEffect, useRef, useState } from "react";
import PlaylistMovies from "../PlaylistMovies";
import LocalStoragePlaylist from "@/models/LocalStoragePlaylist";
import Movie from "@/models/Movie";
import {
  Box,
  Button,
  FormControl,
  FormHelperText,
  Input,
  InputLabel,
  TextField,
} from "@mui/material";

const PlaylistSection = () => {
  const [playlist, setPlaylist] = useState<LocalStoragePlaylist | null>(null);
  const nameInputRef = useRef<HTMLInputElement>(null);
  const descriptionInputRef = useRef<HTMLInputElement>(null);

  useEffect(() => {
    const playlist = LocalStorageService.playlist.get();
    setPlaylist(playlist);
  }, []);

  const handleMovieDeletion = (movie: Movie) => {
    const updatedPlaylist = LocalStorageService.playlist.removeMovie(movie);
    setPlaylist(updatedPlaylist);
  };

  const handleSubmit = (event: React.FormEvent<HTMLFormElement>) => {
    event.preventDefault();

    const name = nameInputRef.current?.value ?? "";
    const description = descriptionInputRef.current?.value ?? "";

    const updatedPlaylist = LocalStorageService.playlist.setMetadata({
      name,
      description,
    });

    setPlaylist(updatedPlaylist);
    console.log(updatedPlaylist);
  };

  return (
    <section>
      {playlist && (
        <>
          <Box
            component="form"
            onSubmit={handleSubmit}
            display="flex"
            flexDirection="column"
            gap="20px"
            alignItems="center"
          >
            <TextField
              inputRef={nameInputRef}
              label="Playlist name"
              fullWidth
              defaultValue={playlist.name}
            />
            <TextField
              inputRef={descriptionInputRef}
              label="Playlist description"
              fullWidth
              defaultValue={playlist.description}
            />
            <Button
              variant="contained"
              type="submit"
              color="primary"
              size="large"
            >
              Save the playlist
            </Button>
          </Box>
```

```
            <PlaylistMovies
              movies={playlist.movies}
              onMovieDeletion={handleMovieDeletion}
            />
          </>
        )}
      </section>
    );
  };

  export default PlaylistSection;
```

## Database

Now we would store the playlist, and we will store it in a database so we need to build it

We will use an ORM (object-relational mapping) manager as Prisma

- First install Prisma

```
npm install prisma --save-dev
```

- Initialise Prisma to use SQLite database technology (many others are supported as MySQL, PostgreSQL…)

```
npx prisma init --datasource-provider sqlite
```

- Let's create the model (schema) for the database and migrate it to the database
    - We want a `Playlist` to have an `id` , a `name`, a `description` and a list of `movie`
    - We want each `Movie` to have an `id` which is the same as tmdb id (so a custom one and not an auto-increment one), and a list of `Playlist` .

Indeed, a movie can appear in multiple playlists, and a playlist has multiple movies so our case is a `many to many` relationship

And let’s manage our model and its Many to Many implicitly by Prisma

Look at the documentation: Prisma

▼ Solution for models

`prisma/schema.prisma`

```
// This is your Prisma schema file,
// learn more about it in the docs: https://pris.ly/d/prisma-schema

generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "sqlite"
  url      = env("DATABASE_URL")
}

model Playlist {
  id          Int     @id @default(autoincrement())
  name        String?
  description String?
  movies      Movie[]
}

model Movie {
  id         Int        @id
  playlists Playlist[]
}
```

- Create the database and tables by migrating the model

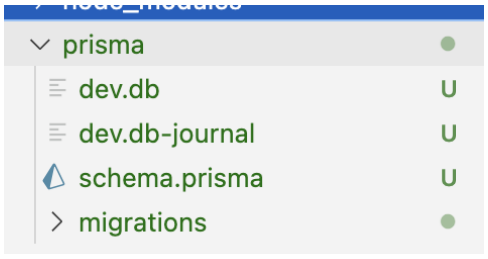
```
npx prisma migrate dev --name init
```

- Prisma provides all the Typescript types by doing a simple command, note that you must rerun it if you update the model, because obviously, the typescript types are based on the Prisma model you define. Run:

```
npx prisma generate
```

You will find your Typescript types there: `node_modules/.prisma/client/index.d.ts`

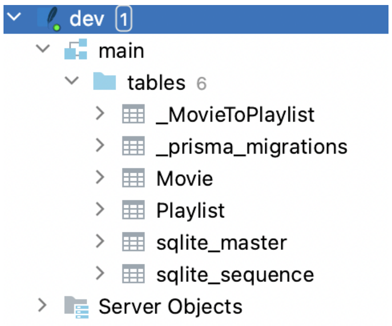
At this point, your project structure should include a Prisma folder



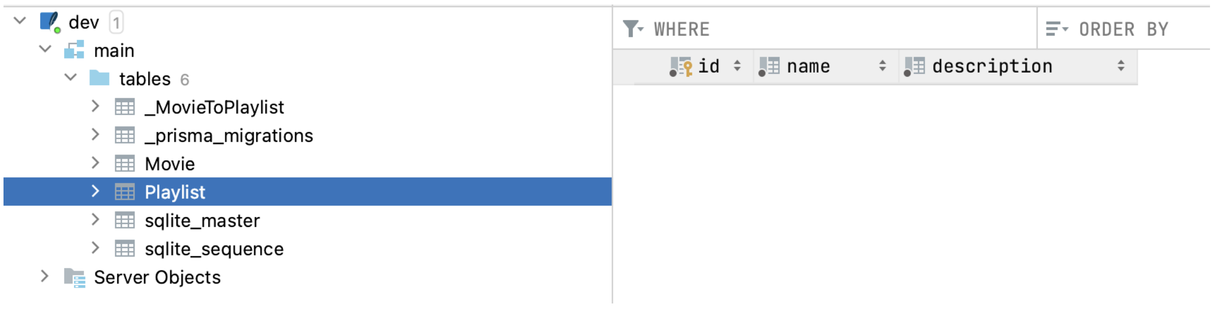
```
prisma
    dev.db            U
    dev.db-journal    U
    schema.prisma     U
  > migrations
```

As we use SQLite, you can see the dev.db file (highlighted one on the screenshot), you can open this database file into an SQL explorer (I personally use DataGrip)

The database should look like this:

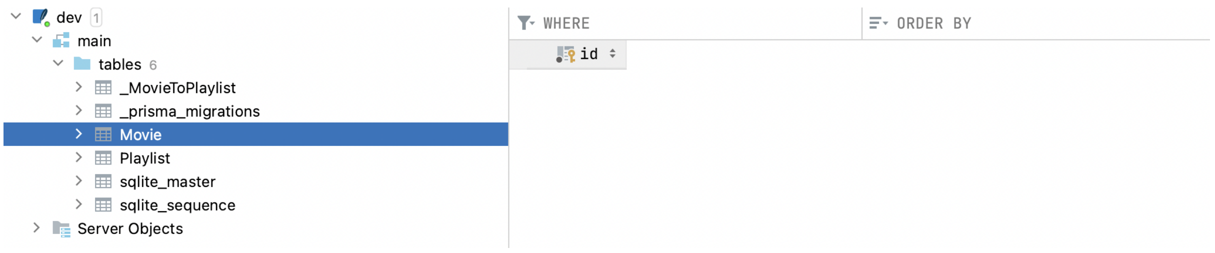


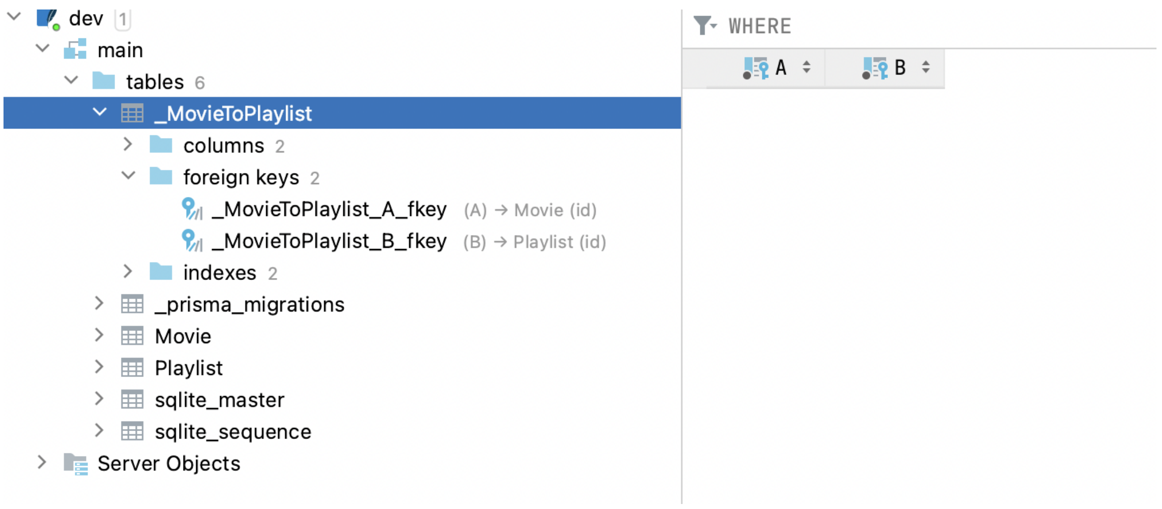
```
dev 1
  main
    tables 6
      > _MovieToPlaylist
      > _prisma_migrations
      > Movie
      > Playlist
      > sqlite_master
      > sqlite_sequence
  > Server Objects
```



```
dev 1
  main
    tables 6
      > _MovieToPlaylist
      > _prisma_migrations
      > Movie
      > Playlist
      > sqlite_master
      > sqlite_sequence
  > Server Objects
```

| id | name | description |
|---|---|---|





## Store the Playlist in the database

Now we want to store the playlist in the database from the function `savePlaylist` (in the playlists/creation page) in which we implemented the console.log of the playlist

This function will be triggered on a click on the button so client side. But the database is obviously on server side, so we need to query the server from the client.

Guess what ? An api call ! A fetch with method POST on the url `http://localhost:3000/api/playlists` with the body as the playlist from local storage

So let’s implement an endpoint to handle that call.

We will do that in the `app/api/playlists/route.ts` file, create this file and check on the documentation how route handlers work https://nextjs.org/docs/app/building-your-application/routing/route-handlers, then focus on https://nextjs.org/docs/app/building-your-application/routing/route-handlers#request-body and simply return the body in response

You can test your endpoint with a query tool like `Postman`

▼ Solution for playlists `POST` endpoint

```
import { NextRequest } from "next/server";

export type CreatePlaylistRequestBody = {
  name: string;
  description: string;
  moviesId: number[];
};

export async function POST(req: NextRequest) {
  const playlist = (await req.json()) as {
    playlist: CreatePlaylistRequestBody;
  };

  return new Response(JSON.stringify(playlist), {
    headers: {
      "Content-Type": "application/json",
    },
    status: 201,
  });
}
```

Then implement the playlist registration to the database

▼ Solution

```
import { PrismaClient } from "@prisma/client";
import { NextRequest } from "next/server";

const prisma = new PrismaClient();

export type CreatePlaylistRequestBody = {
  name: string;
  description: string;
  moviesId: number[];
};

export type CreatePlaylistResponseBody = {
  id: number;
  name: string;
  description: string;
  moviesId: number[];
};

export async function POST(req: NextRequest) {
  const { name, description, moviesId } =
    (await req.json()) as CreatePlaylistRequestBody;

  const playlist = await prisma.playlist.create({
    select: {
      id: true,
      name: true,
      description: true,
      movies: {
        select: {
          id: true,
        },
      },
    },
    data: {
      name,
      description,
      movies: {
        // connect if movie already exists or create it if not
        connectOrCreate: moviesId.map((id) => ({
          where: { id },
          create: { id },
        })),
      },
    },
  });

  return new Response(JSON.stringify(playlist), {
    headers: {
      "Content-Type": "application/json",
    },
    status: 201,
  });
}
```

Now that we have the endpoint, we just have to call it properly from the client,

Let's create another api service as we did for TMDB api: `services/api/application/playlist.ts` and implement your fetch in a function called `create`

▼ Solution

- `services/api/application/playlist.ts`

```
import {
  CreatePlaylistRequestBody,
  CreatePlaylistResponseBody,
} from "@/app/api/playlists/route";
import ApplicationApi from ".";

async function create(
  playlist: CreatePlaylistRequestBody
): Promise<CreatePlaylistResponseBody> {
  return await fetch(`${ApplicationApi.apiUrl}/playlists`, {
    method: "POST",
    body: JSON.stringify(playlist),
  }).then(async (res) => (await res.json()) as CreatePlaylistResponseBody);
}

const playlistApi = {
  create,
};

export default playlistApi;
```

- `services/api/application/index.ts`

```
import playlistApi from "./playlist";

const ApplicationApi = {
  playlist: playlistApi,
  apiUrl: "http://localhost:3000/api",
};

export default ApplicationApi;
```

So turn back to `pages/playlists/creation.ts` and use it in the `savePlaylist` function

Try it and look at the new entries in your database :)

▼ Solution for `PlaylistSection`

```
"use client";

import LocalStorageService from "@/services/localStorage";
import { useEffect, useRef, useState } from "react";
import PlaylistMovies from "../PlaylistMovies";
import LocalStoragePlaylist from "@/models/LocalStoragePlaylist";
import Movie from "@/models/Movie";
import { Box, Button, TextField } from "@mui/material";
import ApplicationApi from "@/services/api/application";

const PlaylistSection = () => {
  const [playlist, setPlaylist] = useState<LocalStoragePlaylist | null>(null);
  const nameInputRef = useRef<HTMLInputElement>(null);
  const descriptionInputRef = useRef<HTMLInputElement>(null);

  useEffect(() => {
    const playlist = LocalStorageService.playlist.get();
    setPlaylist(playlist);
  }, []);

  const handleMovieDeletion = (movie: Movie) => {
    const updatedPlaylist = LocalStorageService.playlist.removeMovie(movie);
    setPlaylist(updatedPlaylist);
  };

  const handleSubmit = async (event: React.FormEvent<HTMLFormElement>) => {
    event.preventDefault();

    const name = nameInputRef.current?.value ?? "";
    const description = descriptionInputRef.current?.value ?? "";

    const updatedPlaylist = LocalStorageService.playlist.setMetadata({
      name,
      description,
    });

    setPlaylist(updatedPlaylist);

    const savedPlaylist = await ApplicationApi.playlist.create({
      name: updatedPlaylist.name,
      description: updatedPlaylist.description,
      moviesId: updatedPlaylist.movies.map((movie) => movie.id),
    });

    console.log(savedPlaylist);
  };

  return (
    <section>
      {playlist && (
        <>
          <Box
            component="form"
            onSubmit={handleSubmit}
            display="flex"
            flexDirection="column"
            gap="20px"
            alignItems="center"
          >
            <TextField
              inputRef={nameInputRef}
              label="Playlist name"
              fullWidth
              defaultValue={playlist.name}
            />
            <TextField
              inputRef={descriptionInputRef}
              label="Playlist description"
              fullWidth
              defaultValue={playlist.description}
            />
            <Button
              variant="contained"
              type="submit"
              color="primary"
              size="large"
            >
```

```
              Save the playlist
            </Button>
          </Box>

          <PlaylistMovies
            movies={playlist.movies}
            onMovieDeletion={handleMovieDeletion}
          />
        </>
      )}
    </section>
  );
};

export default PlaylistSection;
```

Create a new page in charge of displaying the playlist detail `app/playlists/[playlistId]/page.tsx`

This page will just return `<div>Playlist detail of playlist id PLAYLIST_ID</div>`

Once you did this page, turn back to the playlist creation page and add a redirection to `http://localhost:3000/playlists/NEW_CREATED_PLAYLIST_ID` once the playlist is created successfully so the user can review the published playlist and share its link.

▼ Solution for `PlaylistDetailPage` and `PlaylistSection` with redirection

- `app/playlists/[playlistId]/page.tsx`

```
type Props = {
  params: {
    playlistId: string;
  };
};

const PlaylistDetailPage = ({ params }: Props) => {
  return <div>Playlist detail of playlist id {params.playlistId}</div>;
};

export default PlaylistDetailPage;
```

- `components/PlaylistSection/index.tsx`

```
"use client";

import LocalStorageService from "@/services/localStorage";
import { useEffect, useRef, useState } from "react";
import PlaylistMovies from "../PlaylistMovies";
import LocalStoragePlaylist from "@/models/LocalStoragePlaylist";
import Movie from "@/models/Movie";
import { Box, Button, TextField } from "@mui/material";
import ApplicationApi from "@/services/api/application";
import { useRouter } from "next/navigation";

const PlaylistSection = () => {
  const router = useRouter();
  const [playlist, setPlaylist] = useState<LocalStoragePlaylist | null>(null);
  const nameInputRef = useRef<HTMLInputElement>(null);
  const descriptionInputRef = useRef<HTMLInputElement>(null);

  useEffect(() => {
    const playlist = LocalStorageService.playlist.get();
    setPlaylist(playlist);
  }, []);

  const handleMovieDeletion = (movie: Movie) => {
    const updatedPlaylist = LocalStorageService.playlist.removeMovie(movie);
    setPlaylist(updatedPlaylist);
  };

  const handleSubmit = async (event: React.FormEvent<HTMLFormElement>) => {
    event.preventDefault();

    const name = nameInputRef.current?.value ?? "";
    const description = descriptionInputRef.current?.value ?? "";

    const updatedPlaylist = LocalStorageService.playlist.setMetadata({
      name,
      description,
    });

    setPlaylist(updatedPlaylist);

    const savedPlaylist = await ApplicationApi.playlist.create({
      name: updatedPlaylist.name,
      description: updatedPlaylist.description,
      moviesId: updatedPlaylist.movies.map((movie) => movie.id),
    });

    router.push(`/playlists/${savedPlaylist.id}`);
  };

  return (
    <section>
      {playlist && (
        <>
          <Box
            component="form"
            onSubmit={handleSubmit}
            display="flex"
            flexDirection="column"
            gap="20px"
            alignItems="center"
          >
            <TextField
              inputRef={nameInputRef}
              label="Playlist name"
              fullWidth
              defaultValue={playlist.name}
            />
            <TextField
              inputRef={descriptionInputRef}
              label="Playlist description"
              fullWidth
              defaultValue={playlist.description}
            />
            <Button
              variant="contained"
              type="submit"
              color="primary"
              size="large"
            >
              Save the playlist
            </Button>
          </Box>

          <PlaylistMovies
            movies={playlist.movies}
            onMovieDeletion={handleMovieDeletion}
          />
        </>
      )}
    </section>
  );
};

export default PlaylistSection;
```

**Now do `npm run build` and `npm start` to try application and be proud:**

**You complete this workshop. Well done 😎**