TP - Next - App - Router - part 1

We want to create a web application to find films and show their details on click

You can find my implementation alive there: https://ssr-movie-app-next-app-router.rael-calitro.ovh/search

Create the App

Create the application with NextJS

```
npx create-next-app@latest --ts ssr-movie-app
```

Answer all default values by pressing `enter`

[Optional] Setup VSCode debugger

simply create this file at the root of the project: `.vscode/launch.json`

Then put this inside:

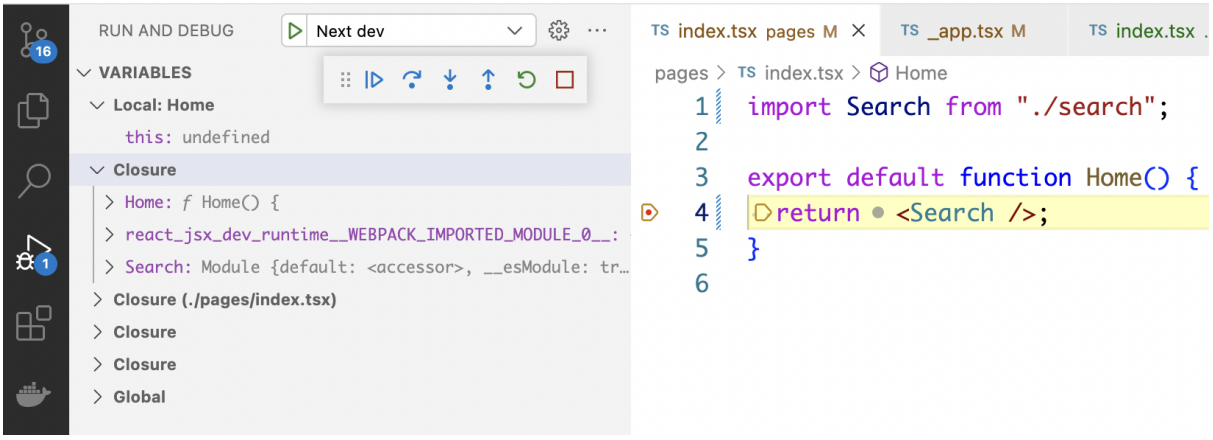
```
{
  // Use IntelliSense to learn about possible attributes.
  // Hover to view descriptions of existing attributes.
  // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
  "version": "0.2.0",
  "configurations": [
    // setup to run and debug into in vscode (debug server code)
    {
      "name": "Next dev",
      "request": "launch",
      "runtimeArgs": ["run-script", "dev"],
      "sourceMaps": true,
      "runtimeExecutable": "npm",
      "cwd": "${workspaceFolder}",
      "skipFiles": ["<node_internals>/**"],
      "type": "node",
      "console": "integratedTerminal"
    }
  ]
}
```

Now you can setup breakpoints (red dot)

```
import Search from "./search";

export default function Home() {
  return <Search />;
}

```

So starting your app with VSC

Will now stop into your code when it’s reached



Try it, request the home page, it should stop on your breakpoint

You can check all the variables on the left menu, evaluate some expression, etc…

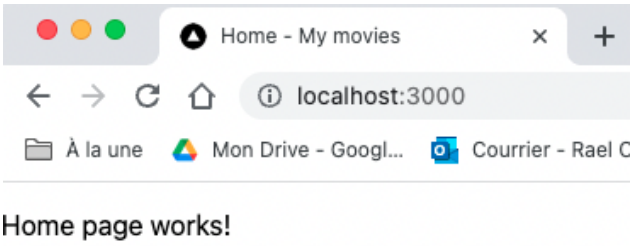
Way better than debugging with console.log()

Set the home page

You’ll already have the Home page created as the app/page.tsx file (Remember that is necessary for your page to be named page.tsx in Next.js)

For now this page should only display “Home page works!” with the page title (in the browser tab):

> Home - my movies



Install Mui (material-ui) properly for Next

We will base our design on Material UI, so install the libraries but since we want to run it also server side. We must do these additonnal steps below.

Install the librairies:

```
npm install @mui/material @emotion/react @emotion/styled @emotion/server @mui/icons-material
```

In order to configure `MUI` for `Next.js` , we need to configure `emotion` cache.

https://mui.com/material-ui/guides/next-js-app-router/

1. create a ThemeRegistry provider `app/ThemeRegistry.tsx`

```
"use client";

import createCache from "@emotion/cache";
import { useServerInsertedHTML } from "next/navigation";
import { CacheProvider } from "@emotion/react";
import { ThemeProvider } from "@mui/material/styles";
import CssBaseline from "@mui/material/CssBaseline";
import { useState } from "react";
import { theme } from "./theme";

// This implementation is from emotion-js
// https://github.com/emotion-js/emotion/issues/2928#issuecomment-1319747902
export default function ThemeRegistry(props: any) {
  const { options, children } = props;

  const [{ cache, flush }] = useState(() => {
    const cache = createCache(options);
    cache.compat = true;
    const prevInsert = cache.insert;
    let inserted: string[] = [];
    cache.insert = (...args) => {
      const serialized = args[1];
      if (cache.inserted[serialized.name] === undefined) {
        inserted.push(serialized.name);
      }
      return prevInsert(...args);
    };
    const flush = () => {
      const prevInserted = inserted;
```

```
      inserted = [];
      return prevInserted;
    };
    return { cache, flush };
  });

  useServerInsertedHTML(() => {
    const names = flush();
    if (names.length === 0) {
      return null;
    }
    let styles = "";
    for (const name of names) {
      styles += cache.inserted[name];
    }
    return (
      <style
        key={cache.key}
        data-emotion={`${cache.key} ${names.join(" ")}`}
        dangerouslySetInnerHTML={{
          __html: options.prepend ? `@layer emotion {${styles}}` : styles,
        }}
      />
    );
  });

  return (
    <CacheProvider value={cache}>
      <ThemeProvider theme={theme}>
        <CssBaseline />
        {children}
      </ThemeProvider>
    </CacheProvider>
  );
}
```

2. Create your custom theme, properties fallback to default MUI values when not set, so let's create an empty theme in file `app/theme.ts`

```
import { createTheme } from "@mui/material";

export const theme = createTheme({});
```

3. Use the `ThemeRegistry` in the `RootLayout` , for this setting follow next step

Build the root layout

In `app/layout.tsx` set this layout

```
export default function RootLayout({
  children,
}: {
  children: React.ReactNode;
}) {
```

```
  return (
    <html lang="en">
      <body className={inter.className}>
        <ThemeRegistry options={{ key: "mui" }}>
          <header>
            <h1>My movies app</h1>
          </header>
          <main>{children}</main>
          <footer>Footer</footer>
        </ThemeRegistry>
      </body>
    </html>
  );
}
```

Note the presence of `ThemeRegistry` wrapping all the app in order to complete the MUI server setting.

The layout should import the `globals.css`

```
html,
body {
  margin: 0;
  box-sizing: border-box;
}

body {
  padding: 0 20px;
  display: flex;
  flex-direction: column;
  min-height: 100vh;
  width: 100vw;
}
```

And a `layout.css`

```
header {
  height: 80px;
}

main {
  flex: 1;
}

footer {
  height: 80px;
}
```

Build the home page content

This page will be our landing page.

We would provide some description about the web site as

```
import { Typography } from "@mui/material";
import { Metadata } from "next";

export const metadata: Metadata = {
  title: "Home - My movies",
  description: "A website about building playlists of movies",
};

export default function Home() {
  return (
    <div>
      <Typography>
        Welcome to this site about building playlists of movies...
      </Typography>
    </div>
  );
}
```

Important: Notice the presence of metadata constant

To be able to interact with the page metadata (the <Head> HTML tag and so on), we need to export a `Metadata` object in a const exactly called `metadata` or a function `generateMetadata` for dynamic metadata settings. And best practices encourage us to provide page description and title.

Documentation https://nextjs.org/docs/app/building-your-application/optimizing/metadata

Build the search page

Create a file for the page `app/search/page.tsx` which returns only a `p`

Why this exact file path ? See routing documentation https://nextjs.org/docs/app/building-your-application/routing

```
<p>Search page works</p>
```

Create a folder “components” and create the search bar component, following this material ui component

```
<TextField
      InputProps={{
        endAdornment: (
          <InputAdornment position="end">
            <SearchIcon onClick={handleSearch} />
          </InputAdornment>
        ),
      }}
      color="primary"
      variant="outlined"
      placeholder="Search..."
      size="small"
      onKeyDown={onSearchKeyDown}
    />
```

Important: As this component handles DOM events which exist only client side (frontend side) obviously, we need to set this component as a client component with the special directive `'use client'`

▼ `SearchBar` component solution

```
"use client";

import { InputAdornment, TextField } from "@mui/material";
import SearchIcon from "./SearchIcon";

const SearchBar = () => {
  const handleSearch = () => { /* we will implement the logic later */ };

  const onSearchKeyDown: KeyboardEventHandler<HTMLDivElement> = (e) => {
    if (e.code !== "Enter") return;

    handleSearch();
  };

  return (
    <TextField
      InputProps={{
        endAdornment: (
          <InputAdornment position="end">
            <SearchIcon onClick={handleSearch} />
          </InputAdornment>
        ),
      }}
      color="primary"
      variant="outlined"
      placeholder="Search..."
      size="small"
      onKeyDown={onSearchKeyDown}
    />
  );
};

export default SearchBar;
```

`SearchIcon` is also a client component as it deals with onClick event, you can nest this component folder into SearchBar component folder

```
import { Search } from "@mui/icons-material";
import classes from "./classes.module.css";

type Props = {
  onClick: () => void;
};

export default function SearchIcon({ onClick }: Props) {
  return (
    <div className={classes.searchIcon} onClick={onClick}>
      <Search />
    </div>
  );
}
```

```
.searchIcon {
  cursor: pointer;
  display: flex;
  align-items: center;
  justify-content: center;
  padding: 4px;
}
```

Use that component in `app/search/page.tsx`

```
import SearchBar from "@/components/SearchBar";
import classes from "./page.module.css";
import { Metadata } from "next";

export const metadata: Metadata = {
  title: "Search - My movies",
  description: "Search for movies",
};

export default function Search() {
  return (
    <div className={classes.root}>
      <SearchBar />
    </div>
  );
}
```

Let’s add navigations link into our header, but before let’s revamp the root layout in order to to split header and footer content to their respective components

```
import type { Metadata } from "next";
import { Inter } from "next/font/google";
import "./globals.css";
import "./layout.css";
import MainHeader from "@/components/MainHeader";
import MainFooter from "@/components/MainFooter";
import ThemeRegistry from "./ThemeRegistry";

const inter = Inter({ subsets: ["latin"] });

export const metadata: Metadata = {
  title: "My movies",
  description: "A playlists of movies application",
};

export default function RootLayout({
  children,
}: {
  children: React.ReactNode;
}) {
  return (
    <html lang="en">
      <body className={inter.className}>
        <ThemeRegistry options={{ key: "mui" }}>
          <header>
```

```
            <MainHeader />
          </header>

          <main>{children}</main>

          <footer>
            <MainFooter />
          </footer>
        </ThemeRegistry>
      </body>
    </html>
  );
}
```

Now let's improve the `MainHeader` adding nav links using Next `<Link>` component

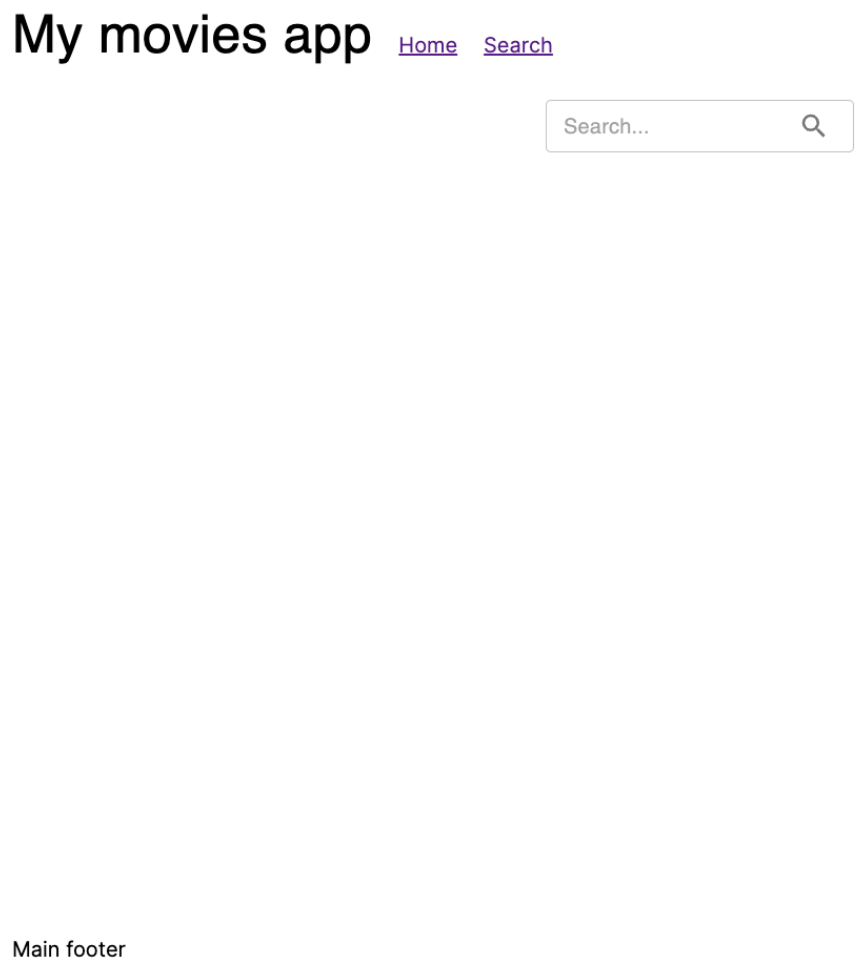
https://nextjs.org/docs/app/api-reference/components/link

```
// app/components/MainHeader/index.tsx

import { Typography } from "@mui/material";
import Link from "next/link";
import classes from "./index.module.css";

const MainHeader = () => {
  return (
    <div className={classes.root}>
      <Typography component="h1" fontSize={"1.8rem"}>
        My movies app
      </Typography>
      <nav className={classes.navigation}>
        <Link href="/">
          Home
        </Link>
        <Link href="/search">
          Search
        </Link>
      </nav>
    </div>
  );
};

export default MainHeader;
```

```
/* app/components/MainHeader/index.module.css */

.root {
  display: flex;
  align-items: baseline;
  gap: 20px;
}

.navigation {
  display: flex;
  gap: 20px;
}
```

Now the home page should look like

And the search page



Build the search bar logic

We want to redirect on `/search/THE_SEARCH_TEXT` page on click on icon or press enter

Once it's done, if you try to click on search icon, you'll be redirected but since we did not create any page for this route, Next replies with a 404, so let's implement the search result page in the next step.

But before let's create a global `not-found` page to fit in our layout

https://nextjs.org/docs/app/api-reference/file-conventions/not-found

```
import Link from "next/link";

const NotFound = () => {
  return (
    <div>
      <p>404 - Page not found </p>
      <Link href="/">Go home</Link>
    </div>
  );
};
```

```
export default NotFound;
```

Now let's implement the logic of handleSearch function into SearchBar Component to navigate on the `/search/SEARCH_TEXT` using `useRouter`

Client navigation documentation https://nextjs.org/docs/app/api-reference/functions/use-router

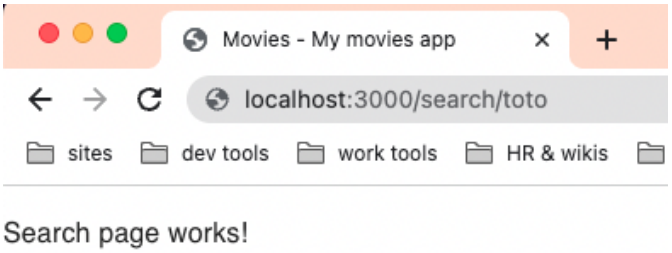
▼ Full `SearchBar` solution

```
"use client";

import { InputAdornment, TextField } from "@mui/material";
import { KeyboardEventHandler, useRef } from "react";
import SearchIcon from "./SearchIcon";
import { useRouter } from "next/navigation";

const SearchBar = () => {
  const inputRef = useRef<HTMLInputElement>(null);
  const router = useRouter();

  const handleSearch = () => {
    const input = inputRef.current;
    if (!input) return;

    const searchText = input.value;
    if (!searchText) return;

    router.push(`/search/${searchText}`);
  };

  const onSearchKeyDown: KeyboardEventHandler<HTMLDivElement> = (e) => {
    if (e.code !== "Enter") return;

    handleSearch();
  };

  return (
    <TextField
      inputRef={inputRef}
      InputProps={{
        endAdornment: (
          <InputAdornment position="end">
            <SearchIcon onClick={handleSearch} />
          </InputAdornment>
        ),
      }}
      color="primary"
      variant="outlined"
      placeholder="Search..."
      size="small"
      onKeyDown={onSearchKeyDown}
    />
  );
};

export default SearchBar;
```

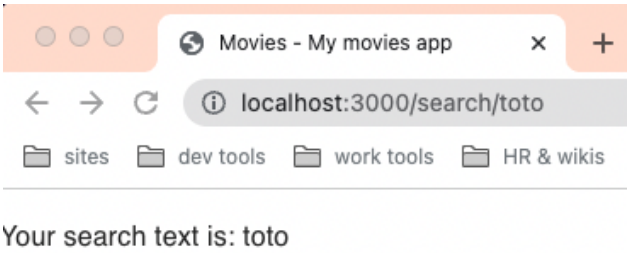
Build the searchText page

Create a file for the route search/[searchText] `app/search/[searchText]/page.tsx`

For now this page just needs to have a proper title (browser tab) and display this when you type `toto` in the search bar and press enter



Let's display something more interesting like the search text, like this



You must get it from props given by server props, don't use the hook useRouter because we will really process it in the server and need fetch the data of the TMDB API from there.

Here is the type of server props, **note that `searchText` matches the slug route name**

Page props documentation https://nextjs.org/docs/app/api-reference/file-conventions/page

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

Once it's done let's implement the actual process: we need to fetch the data from the API TMDB based on search text.

For that,

- Create a file `models/index.ts` for your application models

```
export type SearchMoviesResult = {
  page: number;
  results: Movie[];
  total_pages: number;
  total_results: number;
};

export type Movie = {
  adult: boolean;
  backdrop_path: string;
  genre_ids: number[];
  id: number;
  original_language: string;
  original_title: string;
  overview: string;
  popularity: number;
  poster_path: string;
  release_date: string;
  title: string;
  video: boolean;
  vote_average: number;
  vote_count: number;
};
```

- Create a file `services/api/tmdb/index.ts`

```
import { Movie, SearchMoviesResult } from "../../models";

const rootUrl = "https://api.themoviedb.org/3";
const apiKey = process.env.NEXT_PUBLIC_TMDB_API_KEY;

export async function fetchMovies(search: string): Promise<SearchMoviesResult> {
  return fetch(
    `${rootUrl}/search/movie?api_key=${apiKey}&language=en-US&query=${search}&page=1&include_adult=false`
  ).then(async (res) => {
    if (res.status >= 400) throw new Error();

    return await res.json();
  }) as Promise<SearchMoviesResult>;
}
```

Once you have the searchResult in the Search page, you can log it, but note the `console.log` should log the values in server console, not in the browser one. It’s because the page is built server side

Refresh the page and you should see something like in your server console

```
{
  page: 1,
  results: [
    {
      adult: false,
      backdrop_path: '/lKbuP4xl9xGXuIfJXPOAwBWd6Zt.jpg',
      genre_ids: [Array],
      id: 1181335,
      original_language: 'es',
      original_title: 'Totó',
      overview: "Ada, an 8-year-old girl, longs to remember her late grandmother Totó's voice and fin
rs she can hear her in a different way.",
      popularity: 2.507,
      poster_path: '/2p4RmsY0Fe2NUKRQzg6MAlL8mHL.jpg',
      release_date: '2023-10-20',
      title: 'Totó',
      video: false,
      vote_average: 0,
      vote_count: 0
    },
    {
      adult: false,
      backdrop_path: '/fxYazFVeOCHpHwuqGuiqcCTw162.jpg',
      genre_ids: [Array],
      id: 8392,
      original_language: 'ja',
      original_title: 'となりのトトロ',
      overview: 'Mei, 4 ans, et Satsuki, 10 ans, s’installent à la campagne avec leur père pour se ra
out autour de la maison et, surtout, l’existence d’animaux étranges et merveilleux, les Totoros, avec
dent le retour de leur mère, elles apprennent que sa sortie de l’hôpital a été repoussée. Mei décide
recherchent en vain. Désespérée, Satsuki va finalement demander de l’aide à son voisin Totoro.',
      popularity: 53.5,
      poster_path: '/eEpy8IiR8N0S6mgkdAjDCMlMYQO.jpg',
      release_date: '1988-04-16',
      title: 'Mon voisin Totoro',
      video: false,
      vote_average: 8.071,
      vote_count: 7195
    },
    {
      adult: false,
      backdrop_path: '/q97XywfrS5WbLXZzhM3LN6CytKZ.jpg',
      genre_ids: [Array],
      id: 709123,
      original_language: 'en',
      original_title: 'Toto',
      overview: 'Rosa (Rosa Forlano), a 90-year-old Nonna, falls in love with a Robot while teaching
 software update. TOTO is a sincere ode to traditions like Rosa’s—who’s also the real-life Nonna of f
n times to come.',
      popularity: 0.6,
      poster_path: '/51grp21WqbqwgSXBoXTMIXijeiU.jpg',
      release_date: '2020-05-29'
```


Implement the SearchResult component

As we would display a table handling the movie search results, we need to implement the UI

So let’s create a server component for the table

```
<TableContainer className={classes.root} component={Paper}>
      <Table sx={{ minWidth: 650 }} aria-label="simple table">
        <TableHead>
          <TableRow>
            <TableCell align="center" className={classes.tableCell}>
              ID
```

```
            </TableCell>
            <TableCell align="center" className={classes.tableCell}>
              Titre
            </TableCell>
            <TableCell align="center" className={classes.tableCell}>
              Évaluation
            </TableCell>
            <TableCell align="center" className={classes.tableCell}>
              Nb de votes
            </TableCell>
            <TableCell align="center" className={classes.tableCell}>
              Popularité
            </TableCell>
            <TableCell align="center" className={classes.tableCell}>
              Date de sortie
            </TableCell>
          </TableRow>
        </TableHead>
        <TableBody>
          {results.map((movie) => (
            <TableRow
              key={movie.id}
              sx={{ "&:last-child td, &:last-child th": { border: 0 } }}
              className={classes.row}
              onClick={() => {
                /* handle navigation */
              }}
            >
              <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>
            </TableRow>
          ))}
        </TableBody>
      </Table>
    </TableContainer>
  );
```

Use that component into SearchText page

At this point you should have an issue with the SearchResult component because it handle frontend logic into server as it contains a DOM event handler : `onClick`

In order to handle that properly we need to wrap this event handler into a client component, we can name it `ClientTableRow` as it would be a generic component that handles any onClick in the row to redirect to a target href

```
"use client";

import { Theme } from "@emotion/react";
import { SxProps, TableRow } from "@mui/material";
import { useRouter } from "next/navigation";
import { PropsWithChildren } from "react";

const ClientTableRow = ({
  children,
  sx,
```

```
  className,
  linkHref,
}: PropsWithChildren<{
  sx: SxProps<Theme>;
  className: string;
  linkHref: string;
}>) => {
  const router = useRouter();

  const handleClick = () => {
    router.push(linkHref);
  };

  return (
    <TableRow sx={sx} className={className} onClick={handleClick}>
      {children}
    </TableRow>
  );
};

export default ClientTableRow;
```

We forward props for ui customisation but more important:

- We provide the target href to navigate on when we'll click on a table row
- We provide the `children` to prepare a compound component pattern. Why ? **Because, by default, children of a client component are also client ones.** So the best practice for this kind generic component is to give the possibility to maintain any server or client component wrapped in the component. And this is achievable by `children` usage in compound component pattern.

Server and client composition pattern documentation https://nextjs.org/docs/app/building-your-application/rendering/composition-patterns#supported-pattern-passing-server-components-to-client-components-as-props

▼ `ClientTableRow` component solution

```
"use client";

import { Theme } from "@emotion/react";
import { SxProps, TableRow } from "@mui/material";
import { useRouter } from "next/navigation";
import { PropsWithChildren } from "react";

const ClientTableRow = ({
  children,
  sx,
  className,
  linkHref,
}: PropsWithChildren<{
  sx: SxProps<Theme>;
  className: string;
  linkHref: string;
}>) => {
  const router = useRouter();

  const handleClick = () => {
    router.push(linkHref);
```

```
  };

  return (
    <TableRow sx={sx} className={className} onClick={handleClick}>
      {children}
    </TableRow>
  );
};

export default ClientTableRow;
```

▼ Final `SearchResult` component solution

```
import {
  TableContainer,
  Paper,
  Table,
  TableHead,
  TableRow,
  TableCell,
  TableBody,
} from "@mui/material";
import ClientTableRow from "../ClientTableRow";
import classes from "./index.module.css";
import { fetchMovies } from "@/services/api/tmdb/movies";

type Props = {
  searchText: string;
};

const SearchResult = async ({ searchText }: Props) => {
  const searchMoviesResult = await fetchMovies(searchText);
  const results = searchMoviesResult.results;

  return (
    <TableContainer className={classes.root} component={Paper}>
      <Table sx={{ minWidth: 650 }} aria-label="search movies result 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>
          </TableRow>
        </TableHead>
        <TableBody>
          {results.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>
            </ClientTableRow>
          ))}
        </TableBody>
      </Table>
    </TableContainer>
  );
};

export default SearchResult;
```

```
// index.module.css

.root {
  margin: 20px 0;
}

.headCell {
  font-weight: bold;
}

.row {
  cursor: pointer;
}

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

▼ `SearchText` page solution

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

type Props = {
  params: {
    searchText: string;
  };
};

export const generateMetadata = ({ params }: Props) => {
  return {
    title: `Search: ${params.searchText} - My movies`,
    description: `Search results for ${params.searchText}`,
  };
};

const SearchTextPage = async ({ params }: Props) => {
  return (
    <div>
      <p>Your search text is: {params.searchText}</p>
```

```
      <SearchResult searchText={params.searchText} />
    </div>
  );
};

export default SearchTextPage;
```

Note the presence of `generateMetadata` , exporting a function with this exact name allows us to parameter the page metadata dynamically, and this function takes the same kind of props for page.

Documentation of dynamic metadata: https://nextjs.org/docs/app/building-your-application/optimizing/metadata#dynamic-metadata

At this point, as previously like for the search results, when you click on a movie row in the table, you'll land on the `not-found` page. Why ? Obviously because we did not implement any page for this route, so let's do it.

Build the movie details page

Create a file `app/movies/[movieId]/page.tsx`

Add `fetchMovie` in `services/api/tmdb/index.ts` file

```
export async function fetchMovie(id: number): Promise<Movie> {
  return fetch(`${rootUrl}/movie/${id}?api_key=${apiKey}&language=en-US`).then(
    async (res) => {
      if (res.status >= 400) throw new Error();
      return await res.json();
    }
  ) as Promise<Movie>;
}
```

And use it in the Page, but as the url parameter for movieId could be of any string value and the TMDB api accepts only id as number, we must validate it before calling the api, if id is invalid we can simply return the not-found page

```
import MovieSection from "@/components/MovieSection";
import { Typography } from "@mui/material";
import { notFound } from "next/navigation";
import classes from "./page.module.css";
import { fetchMovie } from "@/services/api/tmdb";
import { Metadata } from "next";

type Props = {
  params: {
    movieId: string;
  };
};

const handleParams = ({ params }: Props) => {
  const movieId = Number(params.movieId);
  const isMovieIdValid = Number.isInteger(movieId);
```

```
  return { movieId, isMovieIdValid };
};

export async function generateMetadata({ params }: Props): Promise<Metadata> {
  const { movieId, isMovieIdValid } = handleParams({ params });
  if (!isMovieIdValid) {
    return notFound();
  }

  const movie = await fetchMovie(movieId);

  return {
    title: `${movie.title} - My movies`,
    description: `Page details of movie: ${movie.title}`,
  };
}

const MoviePage = async ({ params }: Props) => {
  const { movieId, isMovieIdValid } = handleParams({ params });

  if (!isMovieIdValid) {
    return notFound();
  }

  return (
    <div className={classes.root}>
      <Typography>Movie ID: {movieId}</Typography>
      <MovieSection movieId={movieId} />
    </div>
  );
};

export default MoviePage;
```

But let's create a `MovieSection` server component in charge of fetching the movie and provide it to a component in charge of displaying the movie so called `MovieCard`

▼ Solution for `MovieSection`

```
import { fetchMovie } from "@/services/api/tmdb";
import MovieCard from "../MovieCard";

type Props = {
  movieId: number;
};

const MovieSection = async ({ movieId }: Props) => {
  const movie = await fetchMovie(movieId);

  return (
      <section>
        <MovieCard movie={movie} />
      </section>
  );
};

export default MovieSection;
```

Now let's create the `MovieCard` component in charge of displaying the movie details without handling the image for now

▼ Solution for `MovieCard` without handling image

```
// import { getMovieImageUrl } from "@/services/api/tmdb";
import { Card, CardContent, Box, Rating, Typography } from "@mui/material";
import classes from "./index.module.css";
// import Image from "next/image";
import Movie from "@/models/Movie";

type Props = {
  movie: Movie;
};

const MovieCard = async ({ movie }: Props) => {
  // const image = getMovieImageUrl(movie.poster_path);

  return (
    <Card>
      <CardContent className={classes.root}>
        <Box className={classes.imageWrapper}>
          {/* <Image
            priority
            src={image}
            alt={`poster ${movie.title}`}
            width={267}
            height={400}
          /> */}
        </Box>
        <Box className={classes.content}>
          <Rating
            defaultValue={movie.vote_average}
            precision={0.25}
            max={10}
            size="large"
            readOnly
          />
          <Typography gutterBottom variant="h5" component="div" mt={3}>
            {movie.title}
          </Typography>
          <Typography variant="body2" color="text.secondary">
            {movie.overview}
          </Typography>

          <Typography variant="body2" color="text.secondary" mt={3}>
            Date de sortie : {movie.release_date}
          </Typography>
          <Typography variant="body2" color="text.secondary">
            Titre original : {movie.original_title} - VO :{" "}
            {movie.original_language}
          </Typography>
          <Typography variant="body2" color="text.secondary">
            Popularité : {movie.popularity}
          </Typography>
          <Typography variant="body2" color="text.secondary">
            Évaluation : {movie.vote_average}
          </Typography>
          <Typography variant="body2" color="text.secondary">
            Votes : {movie.vote_count}
          </Typography>
        </Box>
      </CardContent>
    </Card>
```

```
  );
};

export default MovieCard;
```

Let’s handle the movie image

1. Add the `retrieveMovieImageUrl` into `services/api/tmdb/index.ts` file

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

const rootUrl = "https://api.themoviedb.org/3";
const apiKey = process.env.TMDB_API_KEY;
export const fallBackMovieImageUrl = "/images/movie-fallback.png";

export async function fetchMovies(search: string): Promise<SearchMoviesResult> {
  return fetch(
    `${rootUrl}/search/movie?api_key=${apiKey}&language=en-US&query=${search}&page=1&include_adult=false`
  ).then(async (res) => {
    if (res.status >= 400) throw new Error();

    return await res.json();
  }) as Promise<SearchMoviesResult>;
}

export async function fetchMovie(id: number): Promise<Movie> {
  return fetch(`${rootUrl}/movie/${id}?api_key=${apiKey}&language=en-US`)
    .then(async (res) => {
      if (res.status >= 400) throw new Error();

      return await res.json();
    }) as Promise<Movie>;
}

export function retrieveMovieImageUrl(path: string | null) {
  if (!path) return fallBackMovieImageUrl;

  const isPathStartingWithSlash = path?.startsWith("/");
  const pathWithoutStartingSlash = isPathStartingWithSlash
    ? path.substring(1)
    : path;

  return `https://image.tmdb.org/t/p/w500/${pathWithoutStartingSlash}`;
}
```

We handle the `null` value because the movie.poster_path value can be null from the TMDB API if no image is available for the movie.

2. Download and put this fallback image into the folder `/public/images`

`movie-fallback.png`

3. Uncomment image parts in `MovieSection`

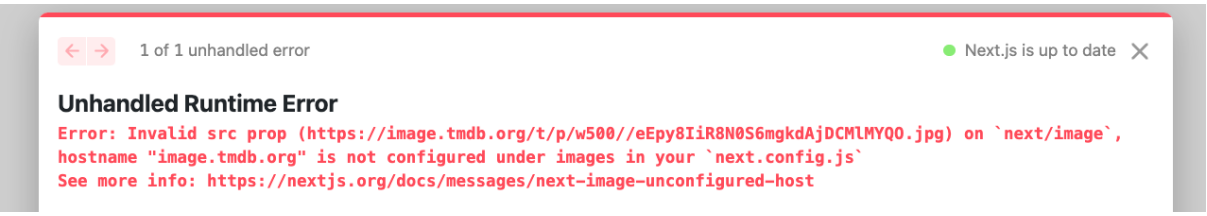
```
import {
  fallBackMovieImageUrl,
  retrieveMovieImageUrl,
} from "@/services/api/tmdb";
import { Card, CardContent, Box, Rating, Typography } from "@mui/material";
import classes from "./index.module.css";
import Image from "next/image";
import Movie from "@/models/Movie";

type Props = {
  movie: Movie;
};

const MovieCard = async ({ movie }: Props) => {
  const image = retrieveMovieImageUrl(movie.poster_path);

  return (
    <Card>
      <CardContent className={classes.root}>
        <Box className={classes.imageWrapper}>
          <Image
            priority
            src={image}
            alt={`poster ${movie.title}`}
            width={267}
            height={400}
            placeholder="blur"
            blurDataURL={fallBackMovieImageUrl}
          />
        </Box>
        <Box className={classes.content}>
          <Rating
            defaultValue={movie.vote_average}
            precision={0.25}
            max={10}
            size="large"
            readOnly
          />
          <Typography gutterBottom variant="h5" component="div" mt={3}>
            {movie.title}
          </Typography>
          <Typography variant="body2" color="text.secondary">
            {movie.overview}
          </Typography>
```

```
          <Typography variant="body2" color="text.secondary" mt={3}>
            Date de sortie : {movie.release_date}
          </Typography>
          <Typography variant="body2" color="text.secondary">
            Titre original : {movie.original_title} - VO :{" "}
            {movie.original_language}
          </Typography>
          <Typography variant="body2" color="text.secondary">
            Popularité : {movie.popularity}
          </Typography>
          <Typography variant="body2" color="text.secondary">
            Évaluation : {movie.vote_average}
          </Typography>
          <Typography variant="body2" color="text.secondary">
            Votes : {movie.vote_count}
          </Typography>
        </Box>
      </CardContent>
    </Card>
  );
};

export default MovieCard;
```

Important: Note the presence of the prop `priority` , it's because this will be our largest contentful paint (LCP) and as it's a measure of Core Web Vitals for SEO score, it's good to optimise it

`<Image>` documentation https://nextjs.org/docs/app/api-reference/components/image

4. At this point, Next alerts you about the image url access

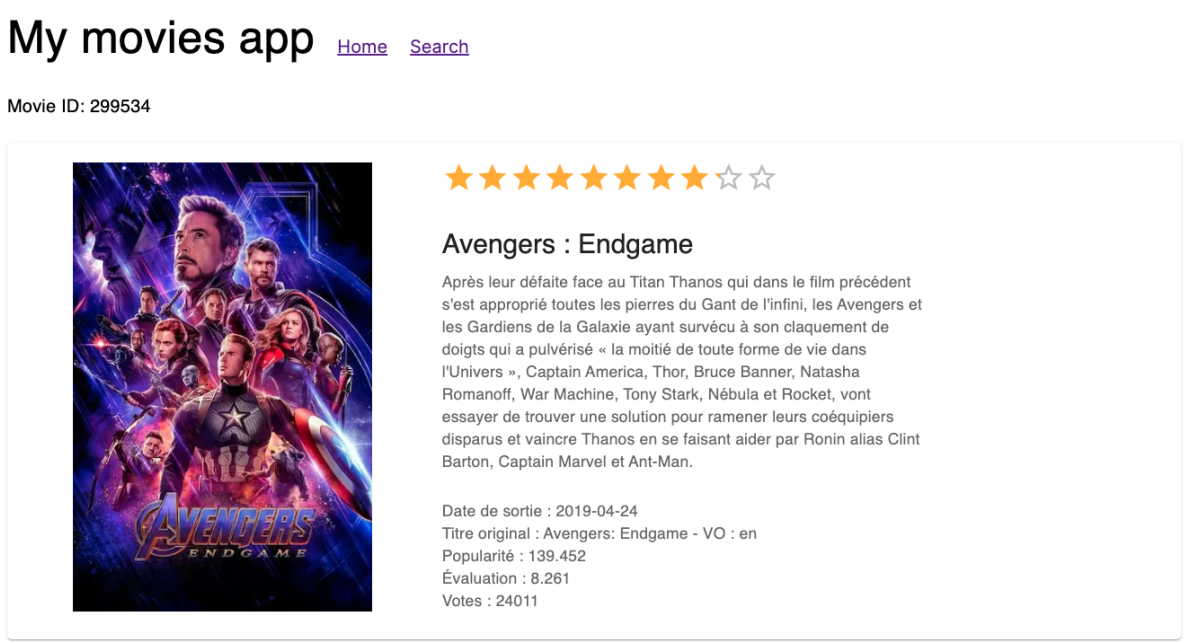


So let's configure Next to be able to access the external image url: update `next.config.js` file

Image Configuration documentation https://nextjs.org/docs/app/api-reference/components/image#configuration-options

```
/** @type {import('next').NextConfig} */
const nextConfig = {
  images: {
    remotePatterns: [
      {
        protocol: "https",
        hostname: "image.tmdb.org",
        port: "",
      },
    ],
  },
};

module.exports = nextConfig;
```

Now, all should be good and the movie detail page should work properly including the movie image



Note: We fetch the movie via the server (backend side as well) instead of frontend side with useEffect, which is really really better for SEO

Now try to search a movie, view its details and go back via the browser navigation, your search results should still be there,

Optimisation by streaming with Suspense

Let’s simulate a slow connexion, we can for example introduce some delay into the api calls

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

const rootUrl = "https://api.themoviedb.org/3";
const apiKey = process.env.TMDB_API_KEY;
export const fallBackMovieImageUrl = "/images/movie-fallback.png";

export async function fetchMovies(search: string): Promise<SearchMoviesResult> {
  return fetch(
    `${rootUrl}/search/movie?api_key=${apiKey}&language=en-US&query=${search}&page=1&include_adult=false`
  ).then(async (res) => {
    await delay(3000);
    if (res.status >= 400) throw new Error();

    return await res.json();
  }) as Promise<SearchMoviesResult>;
```

```
}

export async function fetchMovie(id: number): Promise<Movie> {
  return fetch(`${rootUrl}/movie/${id}?api_key=${apiKey}&language=en-US`).then(
    async (res) => {
      await delay(3000);
      if (res.status >= 400) throw new Error();

      return await res.json();
    }
  ) as Promise<Movie>;
}

export function retrieveMovieImageUrl(path: string | null) {
  if (!path) return fallBackMovieImageUrl;

  const isPathStartingWithSlash = path?.startsWith("/");
  const pathWithoutStartingSlash = isPathStartingWithSlash
    ? path.substring(1)
    : path;

  return `https://image.tmdb.org/t/p/w500/${pathWithoutStartingSlash}`;
}

async function delay(ms: number) {
  return new Promise((resolve) => setTimeout(resolve, ms));
}
```

Notice the addition of `delay` function and its usage into our api calls to introduce a delay of 3 seconds.

Now if we retry the application, for example the search, we will face a delay (during the fetch of movies) before landing on searchText page with result. This is because now it takes at least 3 seconds to the server to build the whole page.

There is a fantastic way to optimise that: `Streaming`

Streaming documentation https://nextjs.org/docs/app/building-your-application/routing/loading-ui-and-streaming

Let’s now wrap what delays the page building into `Suspense`

```
import SearchResult from "@/components/SearchResult";
import { Metadata } from "next";
import { Suspense } from "react";

type Props = {
  params: {
    searchText: string;
  };
};

export function generateMetadata({ params }: Props): Metadata {
  return {
    title: `Search: ${params.searchText} - My movies`,
    description: `Search results for ${params.searchText}`,
  };
}
```

```
const SearchTextPage = async ({ params }: Props) => {
  return (
    <div>
      <p>Your search text is: {params.searchText}</p>
      <Suspense fallback={<div>Loading...</div>}>
        <SearchResult searchText={params.searchText} />
      </Suspense>
    </div>
  );
};

export default SearchTextPage;
```

We now will see the page **immediately** displaying `<p>Your search text is: {params.searchText}</p>` and a `<div>Loading...</div>` during the loading of results

To recap, by just adding the `Suspense` boundary, we are now able to display any part of the page which is wrapped in `Suspense` asynchronously ! So the user could interact with any loaded part of the page even if the whole is not ready ! That we improve so much user experience !

But then… what's the impact on the SEO ?? None because streaming is server rendered as the documentation says https://nextjs.org/docs/app/building-your-application/routing/loading-ui-and-streaming#seo

A very interesting source to deeply understand SEO considerations in website rendering

English version https://web.dev/articles/rendering-on-the-web?hl=en#seo-considerations

French version https://web.dev/articles/rendering-on-the-web?hl=fr#seo-considerations

Let's do the same improvement for the movie page

Wrap the async server component `MovieSection` into `Suspense` boundary

```
import MovieSection from "@/components/MovieSection";
import { Typography } from "@mui/material";
import { notFound } from "next/navigation";
import classes from "./page.module.css";
import { fetchMovie } from "@/services/api/tmdb";
import { Suspense } from "react";
import { Metadata } from "next";

type Props = {
  params: {
    movieId: string;
  };
};

const handleParams = ({ params }: Props) => {
```

```
  const movieId = Number(params.movieId);
  const isMovieIdValid = Number.isInteger(movieId);

  return { movieId, isMovieIdValid };
};

export async function generateMetadata({ params }: Props): Promise<Metadata> {
  const { movieId, isMovieIdValid } = handleParams({ params });
  if (!isMovieIdValid) {
    return notFound();
  }

  const movie = await fetchMovie(movieId);

  return {
    title: `${movie.title} - My movies`,
    description: `Page details of movie: ${movie.title}`,
  };
}

const MoviePage = async ({ params }: Props) => {
  const { movieId, isMovieIdValid } = handleParams({ params });

  if (!isMovieIdValid) {
    return notFound();
  }

  return (
    <div className={classes.root}>
      <Typography>Movie ID: {movieId}</Typography>
      <Suspense fallback={<div>Loading...</div>}>
        <MovieSection movieId={movieId} />
      </Suspense>
    </div>
  );
};

export default MoviePage;
```

Retry to load the page, but we will not trigger the suspense as previously for `SearchText` page. What's happening, before just wrapping with `Suspense` did the trick !

The difference is: here we are depending on a asynchronous `generateMetadata` , indeed, we want to populate the document title with the movie title so we need to fetch it before, and only when it's done the page metadata are ready.

And as the documentation says, generating metadata blocks the page rendering because a ready `<head>` is needed before rendering the page: https://nextjs.org/docs/app/building-your-application/routing/loading-ui-and-streaming#seo

So what ?? How could we customise the document title ?

Create a `ClientHead` to deal with document's `<head>` frontend side

1. In the movie page change the `generateMetadata` to be synchronous so it will not block the page rendering. Then instead of movie title, simply use movieId from route params

```
import MovieSection from "@/components/MovieSection";
import { Typography } from "@mui/material";
import { notFound } from "next/navigation";
import classes from "./page.module.css";
import { Suspense } from "react";
import { Metadata } from "next";

type Props = {
  params: {
    movieId: string;
  };
};

const handleParams = ({ params }: Props) => {
  const movieId = Number(params.movieId);
  const isMovieIdValid = Number.isInteger(movieId);

  return { movieId, isMovieIdValid };
};

export async function generateMetadata({ params }: Props): Promise<Metadata> {
  const { movieId, isMovieIdValid } = handleParams({ params });
  if (!isMovieIdValid) {
    return notFound();
  }

  return {
    title: `${movieId} - My movies`,
    description: `Page details of movie: ${movieId}`,
  };
}

const MoviePage = async ({ params }: Props) => {
  const { movieId, isMovieIdValid } = handleParams({ params });

  if (!isMovieIdValid) {
    return notFound();
  }

  return (
    <div className={classes.root}>
      <Typography>Movie ID: {movieId}</Typography>
      <Suspense fallback={<div>Loading...</div>}>
        <MovieSection movieId={movieId} />
      </Suspense>
    </div>
  );
};

export default MoviePage;
```

2. Create the `ClientHead` component

```
"use client";

import { useEffect } from "react";

type Props = {
  title: string;
  description: string;
};

const ClientHead = ({ title, description }: Props) => {
  useEffect(() => {
    document.title = title;
    document
      .querySelector("meta[name=description]")
      ?.setAttribute("content", description);
  }, [title, description]);

  return <></>;
};

export default ClientHead;
```

3. Create a `PageHead.tsx` file beside the `page.tsx` file in `app/movies/[movieId]` , this server component will use the `ClientHead` component

```
import ClientHead from "@/components/ClientHead";
import { fetchMovie } from "@/services/api/tmdb";

type Props = {
  movieId: number;
};

const PageHead = async ({ movieId }: Props) => {
  const movie = await fetchMovie(movieId);

  return (
    <ClientHead
      title={`${movie.title} - My movies`}
      description={`Page details of movie: ${movie.title}`}
    />
  );
};

export default PageHead;
```

4. Use the `PageHead` component in the `MoviePage`

```
import MovieSection from "@/components/MovieSection";
import { Typography } from "@mui/material";
import { notFound } from "next/navigation";
import classes from "./page.module.css";
import { Suspense } from "react";
import { Metadata } from "next";
```

```
import PageHead from "./PageHead";

type Props = {
  params: {
    movieId: string;
  };
};

const handleParams = ({ params }: Props) => {
  const movieId = Number(params.movieId);
  const isMovieIdValid = Number.isInteger(movieId);

  return { movieId, isMovieIdValid };
};

export async function generateMetadata({ params }: Props): Promise<Metadata> {
  const { movieId, isMovieIdValid } = handleParams({ params });
  if (!isMovieIdValid) {
    return notFound();
  }

  return {
    title: `${movieId} - My movies`,
    description: `Page details of movie: ${movieId}`,
  };
}

const MoviePage = async ({ params }: Props) => {
  const { movieId, isMovieIdValid } = handleParams({ params });

  if (!isMovieIdValid) {
    return notFound();
  }

  return (
    <>
      <Suspense>
        <PageHead movieId={movieId} />
      </Suspense>

      <div className={classes.root}>
        <Typography>Movie ID: {movieId}</Typography>
        <Suspense fallback={<div>Loading...</div>}>
          <MovieSection movieId={movieId} />
        </Suspense>
      </div>
    </>
  );
};

export default MoviePage;
```

Now all should work properly, we should see the `Loading…` text during the movie fetch and the title of the page as movie id fallback. Once the fetch is done we should see the movie content and the page title as the movie title.

We made it 🥳 !!

Build and run your app as production app

Build your app `npm run build` , start it `npm start` , check how fast and almost instant your navigation is. Indeed, already fetched results are cached so they’re shown almost instantly after the first fetch.

You complete this workshop. Well done 😎

Bonus

- Implement the movies sorting on click on table header in search results

This is my implementation:

https://ssr-movie-app-next-app-router.rael-calitro.ovh/