🎥

TP 1

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

- Create the app
- Create the home, details and not found pages
- Add a search bar in the home page that shows result on press enter or click on submit
- Redirect to details page on click on a movie row
- Add a go home button and a friendly card to display the movie details
- Use context to provide the searched text to all our pages so going back and forth doesn’t reset the search results

Create an API key for TMDB

https://developer.themoviedb.org/docs

Create the App

Create the application with create React app

```
npx create-react-app csr-movie-app --template typescript
```

Create the home, details and not found pages

Create folder src/Pages/

Then create the folders HomePage, DetailsPage, NoPage

Inside we will create the index.tsx file containing the implementations

For now, they just have to return <p>Home page works!</p>, <p>Details page works!</p>, <p>Page not found!</p>

Now that we have our pages, we need to implement the routing in order to display them based on the url

So first:

Install the routing library react-router-dom

```
npm install react-router-dom
```

Install types for typescript

```
npm install --save @types/react-router-dom
```

Then, implement the routing in the src/App.tsx

Navigate to the pages trying this urls:

http://localhost:3000 → HomePage

http://localhost:3000/search/ → HomePage

http://localhost:3000/search/foo → NoPage

http://localhost:3000/bar/ → NoPage

http://localhost:3000/details/ → NoPage

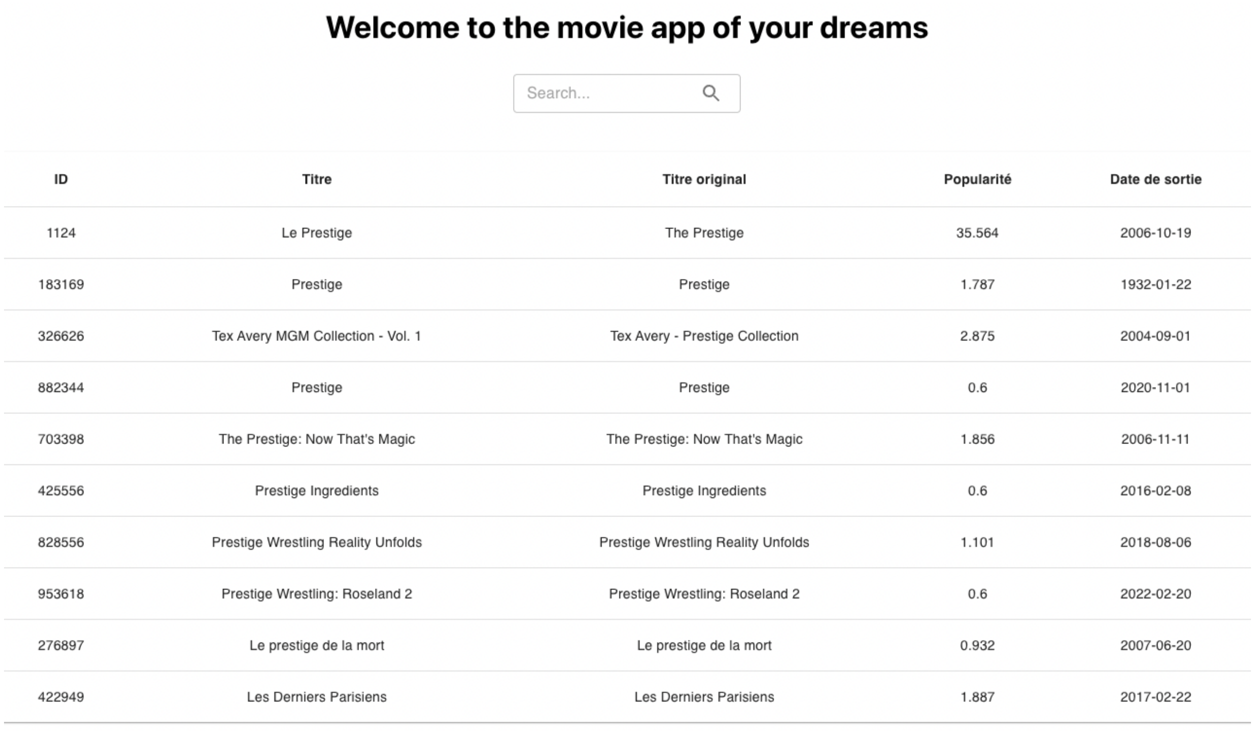
http://localhost:3000/details/foo → DetailsPage

http://localhost:3000/details/bar → DetailsPage

http://localhost:3000/details/bar/foo → NoPage

Add a search bar in the home page that shows result on press enter or click on submit (icon)

Here we want to actually implement HomePage, indeed we want it to look at that:


Welcome to the movie app of your dreams

ID	Titre	Titre original	Popularité	Date de sortie
1124	Le Prestige	The Prestige	35.564	2006-10-19
183169	Prestige	Prestige	1.787	1932-01-22
326626	Tex Avery MGM Collection - Vol. 1	Tex Avery - Prestige Collection	2.875	2004-09-01
882344	Prestige	Prestige	0.6	2020-11-01
703398	The Prestige: Now That's Magic	The Prestige: Now That's Magic	1.856	2006-11-11
425556	Prestige Ingredients	Prestige Ingredients	0.6	2016-02-08
828556	Prestige Wrestling Reality Unfolds	Prestige Wrestling Reality Unfolds	1.101	2018-08-06
953618	Prestige Wrestling: Roseland 2	Prestige Wrestling: Roseland 2	0.6	2022-02-20
276897	Le prestige de la mort	Le prestige de la mort	0.932	2007-06-20
422949	Les Derniers Parisiens	Les Derniers Parisiens	1.887	2017-02-22

We will base our design on Material UI, so install the library:

```
npm install @mui/material @emotion/react @emotion/styled
```

```
npm install @mui/icons-material
```

We want to override styles of MUI by simply passing css classes to the components, for that we need to install:

```
npm install @mui/styled-engine-sc styled-components
```

And update the src/App.tsx as below:

Add this import

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

And wrap all of your tags in

```
<StyledEngineProvider injectFirst>
  ...
</StyledEngineProvider>
```

Redirect to details page on click on a movie row

When we click on a table row result, we want to see the movie details.

So we must redirect to details page based on the movie id.

React-router-dom provides a hook: useNavigate(), that accepts a url, so let's say details/{movie.id} but also a state in which we can put the movie itself so after the redirection to details page, we still have the movie.

For now, just implement this redirection and log the state from details page when we are redirected

Add a go home button and a friendly card to display the movie details

Add a go home button in details and not found pages

We need to keep the movie from the navigation state if it exists, otherwise fetch it by id, but if the response of this fetching is an error then show that the movie is not found

Use context to provide the searched text to all our pages so going back and forth doesn't reset the

search results

For now, if we go back to home we lose the result of the search, it’s because the searched text is reset, an idea would be to use a context to save this searched text at the level of the application itself so we could share it to home and details pages

So add a file src/contexts/SearchTextContextProvider/index.tsx

Then implement the logic in it, create a context with React.createContext and wrap it into the context provider:

```
export const SearchTextContext = React.createContext(...);

function SearchTextContextProvider(props: {children: React.ReactNode}){
  ...
  return (
    <SearchTextContext.Provider value={...}>
      {children}
    </SearchTextContext.Provider>
  )
}
```

Finally use it in src/App.tsx

```
function App() {
  return (
    <StyledEngineProvider injectFirst>
      <SearchTextContextProvider>
        ...
      </SearchTextContextProvider>
    </StyledEngineProvider>
  );
}
```

Now all the children wrapped in SearchTextContextProvider are able to access the searchText

Update your components and pages so they use it

Now try to search a movie, view its details and go back to home, your search results should still be there

You complete this workshop. Well done 😎

Bonus

- Implement the movies sorting on click on table header in search results
- Optimise search result and details queries so they are not always and always re-triggered, you can use React-query library: https://www.npmjs.com/package/react-query

This is my implementation:

https://csr-movie-app.rael-calitro.ovh/

Tools

The 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;
};
```

The function fetch movies

```
export async function fetchMovies(search: string): Promise<SearchResult> {
  return fetch(
    `${baseUrl}/search/movie?api_key=${apiKey}&language=fr-FR&query=${search}&page=1&include_adult=false`
  ).then(async (res) => await res.json()) as Promise<SearchResult>;
}
```

The movies 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>
          {searchResult?.results.map((movie) => (
            <TableRow
              key={movie.id}
              sx={{ "&:last-child td, &:last-child th": { border: 0 } }}
              className={classes.row}
              onClick={()=>{}}
            >
              <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>
```

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

.headCell {
  font-weight: bold;
}

.row {
  cursor: pointer;
}

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

The movie card

```
<Card className={classes.root}>
      <CardMedia
        component="img"
        height="400"
        sx={{ width: "300px", objectFit: "contain" }}
        image={moviePoster}
        alt={movie.title}
        onError={() => setMoviePoster(FallbackImage)}
      />

      <CardContent>
        <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>
      </CardContent>
    </Card
```

```
.root {
  display: flex;
  justify-content: flex-start;
  align-items: center;
  padding: 20px;
  width: 800px;
}
```

Image movie fallback