**LN10: Advanced todo App**

# Contents

[2. Requirement & dependencies 2](#_Toc152490601)

[3. Create route 2](#_Toc152490602)

[4. PrivateRoute component 3](#_Toc152490603)

[5. Login page 3](#_Toc152490604)

[6. Run json-server 7](#_Toc152490605)

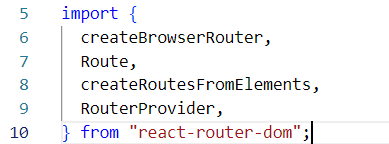
[7. Todo page 8](#_Toc152490606)

# Requirement & dependencies

* 1. **Requirement**
* Latest todo app version of LN09
  1. **Dependencies**
* react-hook-form to handle form in react.
* react-router-dom to handle route in react app.
  1. **Global library**
* json-server to run a server with json file

# Create route

* 1. First, we import those below from react-router-dom in app.js

****

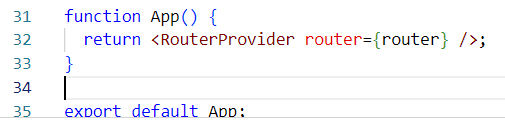
* 1. Second, we create router from createBrowserRouter with routes make from createRoutesFromElements.

****

Then, we insert Route element into createRoutesFromElements. The parent Route element contain the very first path of the route or the url. Next, we have a Login page component in the index route which retains its route as ‘/’ and the TodoApp page component as the path *todo* so its route or url is ‘/todo’. Moreover, it is wrapped by a route with PrivateRoute component, so that only logined user can access it; we will discuss this component later on.

In addition, if the user logined,then they would be navigated to todo page.

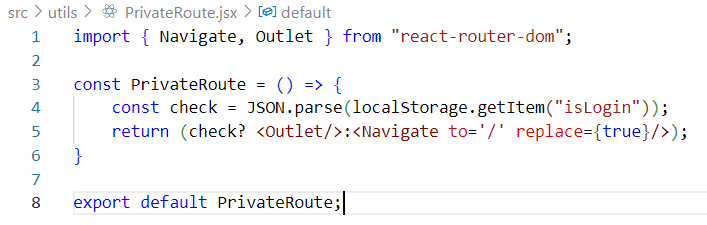
* 1. Finally, we set the router into the RouterProvider and return the RouterProvider.

****

# PrivateRoute component

We create an utils folder contain this Component.

The login logic maybe stores some jwt for checking authorization. In this demonstration, we only use an isLogin flag to check the authorization.

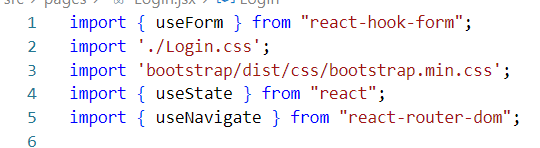


Outlet is used to render child route which is TodoApp in this case.

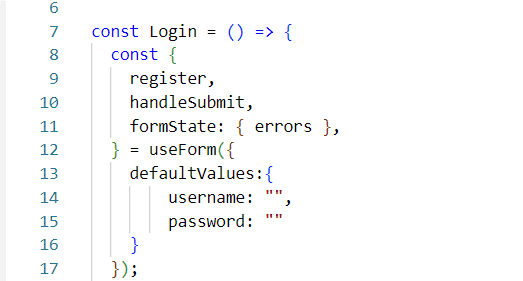
So the return checks if the user login, it will render TodoApp; if not, it will navigate to index page which is login page.

# Login page

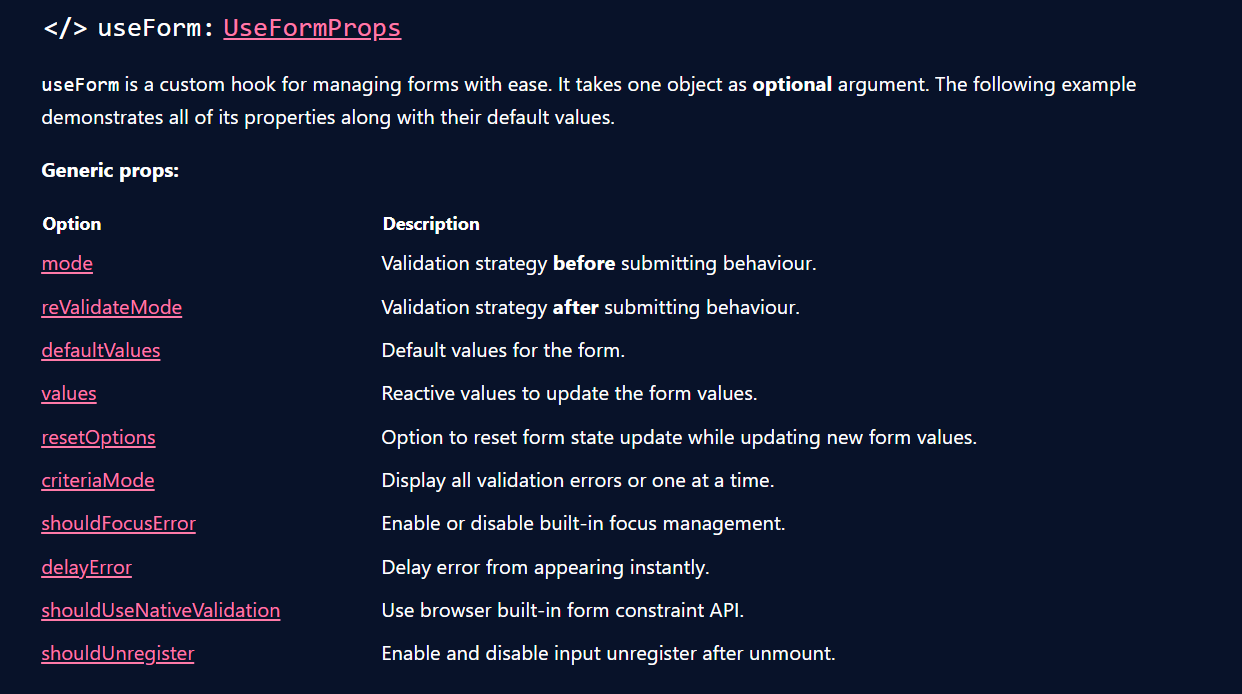
* 1. **Import**

****

* 1. **Use useForm**

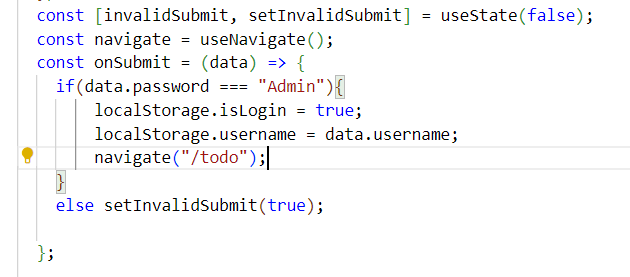


We use useForm and pass an optional object:



Then, we destrucsture the return value to get its props. In this case, we use register, handleSubmit and errors

* 1. **Submit logic**

****

We create an onSubmit function, if password is Admin then the user login and navigate to todo page. The invalidSubmit is used to display some information about it.

* 1. **Jsx form**

****

First, we set the onSubmit attribute is handleSubmit(onSubmit); the handleSubmit takes responsibility to get the form data and pass it to onSubmit function.

Next, in the input tag, the register returns some props to become the input attribute



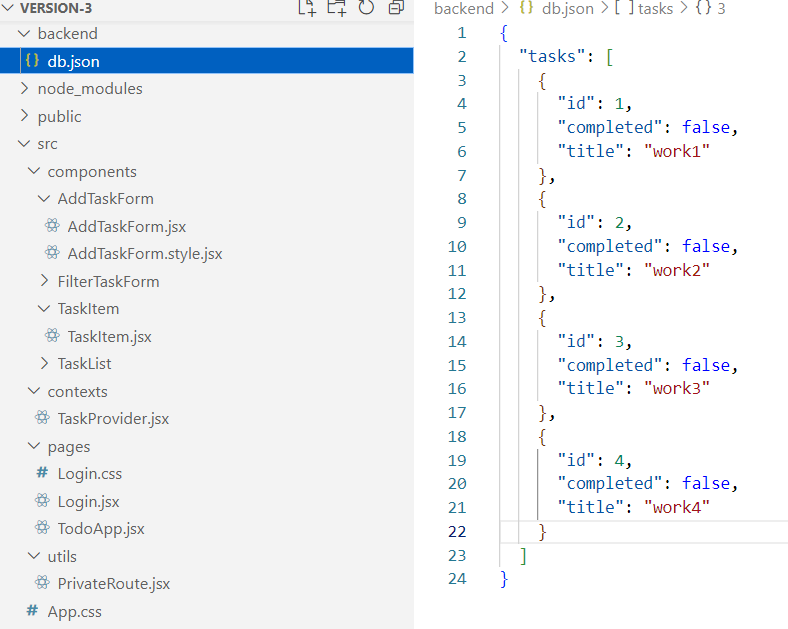
The require in the option object can be true or if you want to display a message, you can pass a string as a message.

After that, we use the errors in the useForm to display the error message.

Finally, the same thing happen in the next input tag, but this time, I but a custom validation.

# Run json-server

We create a folder called backend with a db.json file



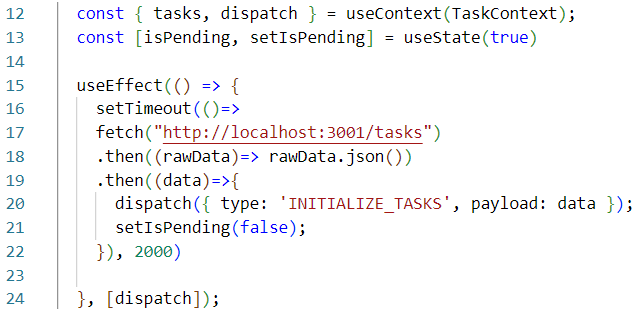
In the json file, each first level object represents a table as RDB or a document in NoSQL.

Then, we run the command to start the server:

json-server --watch .\backend\db.json --port 3001

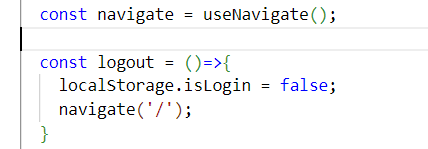
# Todo page

* 1. **Call api in TodoApp component**

****

We change from using localStorage to call api like this. We use setTimeout to present the loading. The isPending to display the pending status.

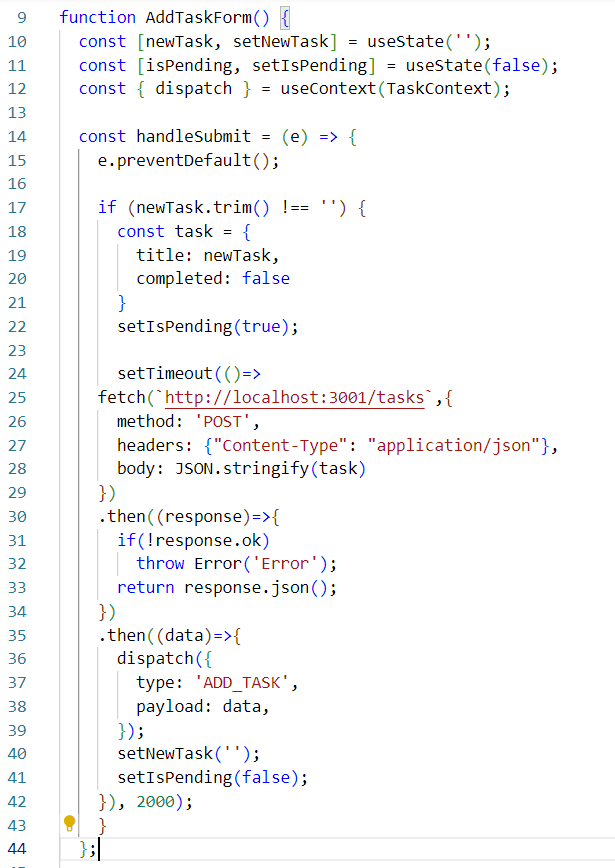
Also, we add a logout button





* 1. **Call Api in add task**

The same way to change from localStorage to call api in this

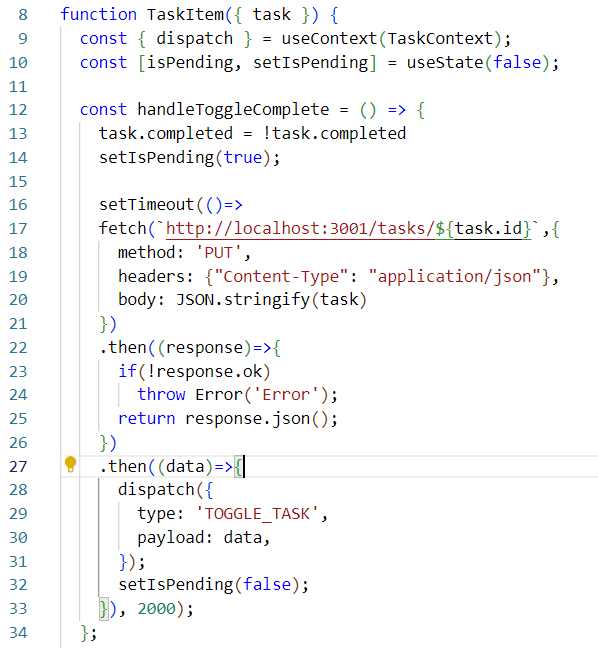


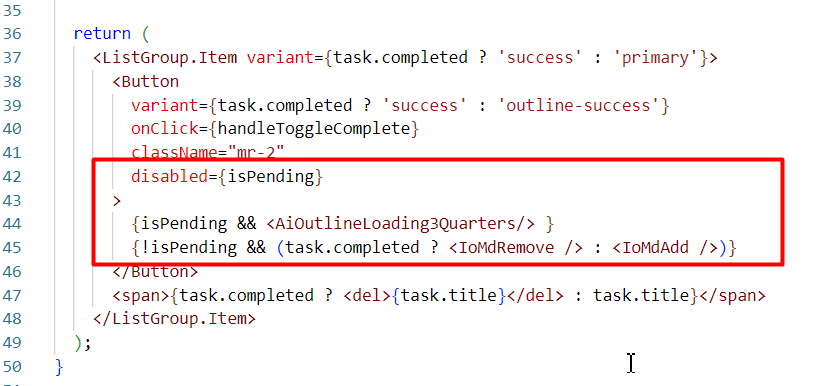
And change the way the button display when pending



* 1. **Call api in change task**

The whole same thing happens in the toggle logic, you can change to call a function to make thing better.





* 1. **Change the reducer logic from the previous version**

