LN10: Advanced todo App

Contents

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

2. **Create route**
    a. First, we import those below from react-router-dom in app.js

```
5   import {
6     createBrowserRouter,
7     Route,
8     createRoutesFromElements,
9     RouterProvider,
10  } from "react-router-dom";
```

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

```
14  const isLogin = JSON.parse(localStorage.getItem("isLogin"));
15
16  const router = createBrowserRouter(
17    createRoutesFromElements(
18      <Route path="/">
19        <Route
20          index
21          element={isLogin ? <Navigate to="/todo" replace={true} /> : <Login />}
22        />
23        <Route element={<PrivateRoute />}>
24          <Route
25            path="/todo"
26            element={
27              <TaskProvider>
28                <TodoApp />
29              </TaskProvider>
30            }
31          />
32        </Route>
33      </Route>
34    )
35  );
```

    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.

c. Finally, we set the router into the RouterProvider and return the RouterProvider.

```
31  function App() {
32    return <RouterProvider router={router} />;
33  }
34
35  export default App;
```

3. 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.

src > utils > PrivateRoute.jsx > default

```
1  import { Navigate, Outlet } from "react-router-dom";
2
3  const PrivateRoute = () => {
4      const check = JSON.parse(localStorage.getItem("isLogin"));
5      return (check? <Outlet/>:<Navigate to='/' replace={true}/>);
6  }
7
8  export default PrivateRoute;
```

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.

4. Login page

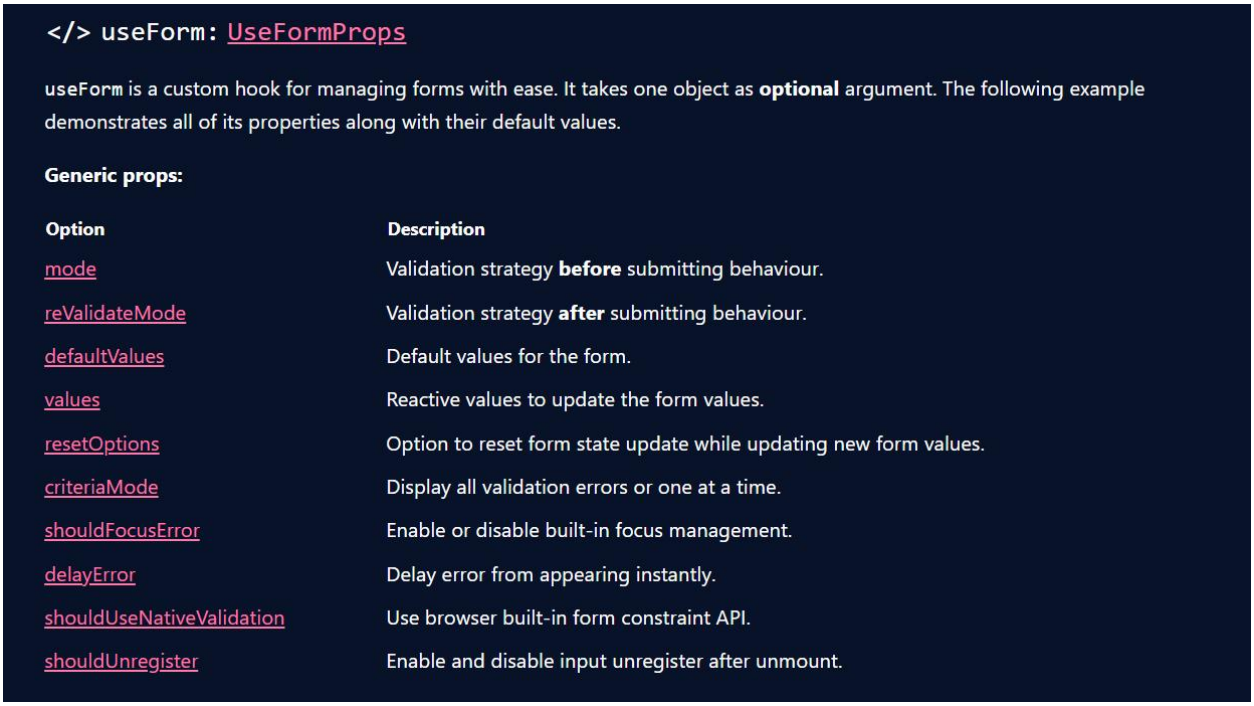
a. Import

```jsx
import { useForm } from "react-hook-form";
import './Login.css';
import 'bootstrap/dist/css/bootstrap.min.css';
import { useState } from "react";
import { useNavigate } from "react-router-dom";

```

b. Use useForm

```jsx

const Login = () => {
  const {
    register,
    handleSubmit,
    formState: { errors },
  } = useForm({
    defaultValues:{
        username: "",
        password: ""
    }
  });
```

We use useForm and pass an optional object:



`</> useForm: UseFormProps`

useForm is a custom hook for managing forms with ease. It takes one object as **optional** argument. The following example demonstrates all of its properties along with their default values.

Generic props:

Option	Description
mode	Validation strategy **before** submitting behaviour.
reValidateMode	Validation strategy **after** submitting behaviour.
defaultValues	Default values for the form.
values	Reactive values to update the form values.
resetOptions	Option to reset form state update while updating new form values.
criteriaMode	Display all validation errors or one at a time.
shouldFocusError	Enable or disable built-in focus management.
delayError	Delay error from appearing instantly.
shouldUseNativeValidation	Use browser built-in form constraint API.
shouldUnregister	Enable and disable input unregister after unmount.

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

c. **Submit logic**

```
const [invalidSubmit, setInvalidSubmit] = useState(false);
const navigate = useNavigate();
const onSubmit = (data) => {
  if(data.password === "Admin"){
      localStorage.isLogin = true;
      localStorage.username = data.username;
      navigate("/todo");
  }
  else setInvalidSubmit(true);

};
```

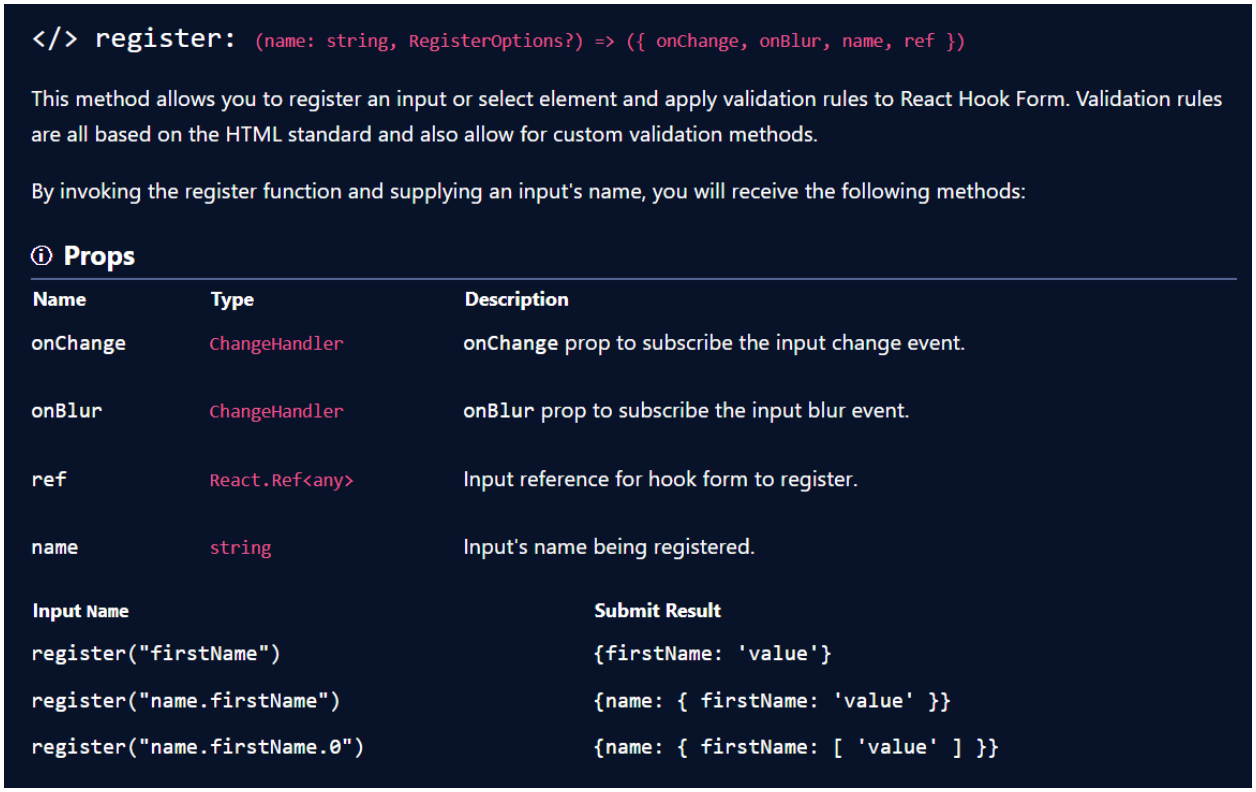
We create an onSubmit function, if password is Admin then the user login and navigate to todo page. The invalidSubmit is used to display some information about it.

d. **Jsx form**

```
30    return (
31      <div className="login">
32        <form onSubmit={handleSubmit(onSubmit)}>
33          <input
34            {...register("username", { required: "Username is required" })}
35            placeholder="Username"
36          />
37          <p>{errors.username?.message}</p>
38          <input
39            {...register("password", {
40              required: "Password is required",
41              validate: {//custom validate
42                notLessThanFour: (v) =>
43                  v.length >= 4 || "Password must has at least 4 characters",
44              },
45            })}
46            placeholder="Password"
47            type="password"
48          />
49          <p>{errors.password?.message}</p>
50          <input type="submit"/>
51        </form>
52        {invalidSubmit && <div>Invalid username & password</div>}
53      </div>
54    );
55  };
56
57  export default Login;
```

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



</> register: (name: string, RegisterOptions?) => ({ onChange, onBlur, name, ref })

This method allows you to register an input or select element and apply validation rules to React Hook Form. Validation rules are all based on the HTML standard and also allow for custom validation methods.

By invoking the register function and supplying an input's name, you will receive the following methods:

ⓘ Props

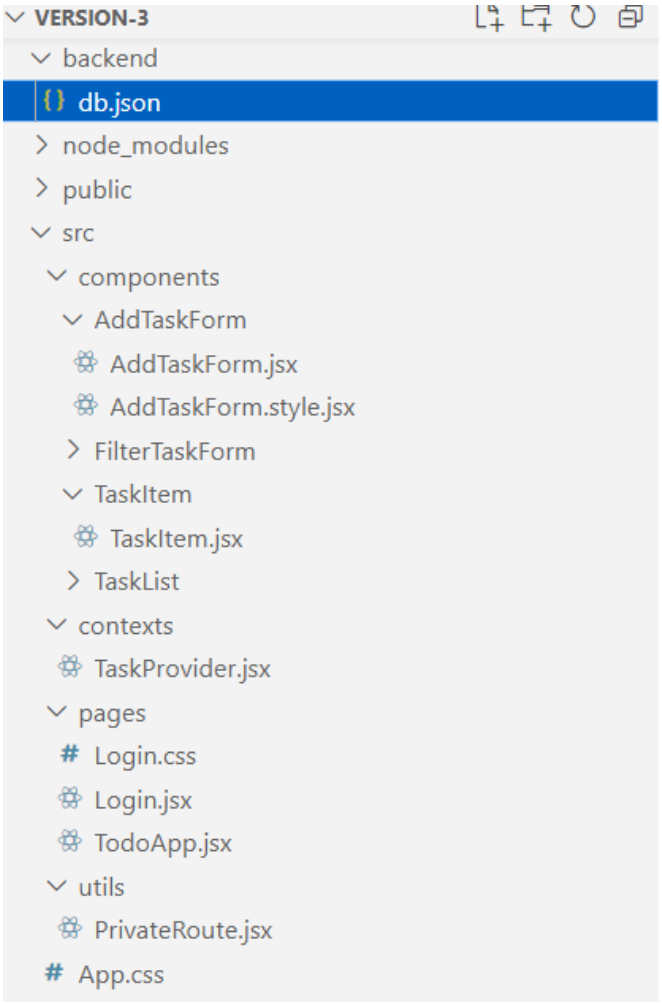
Name	Type	Description
onChange	ChangeHandler	onChange prop to subscribe the input change event.
onBlur	ChangeHandler	onBlur prop to subscribe the input blur event.
ref	React.Ref<any>	Input reference for hook form to register.
name	string	Input's name being registered.

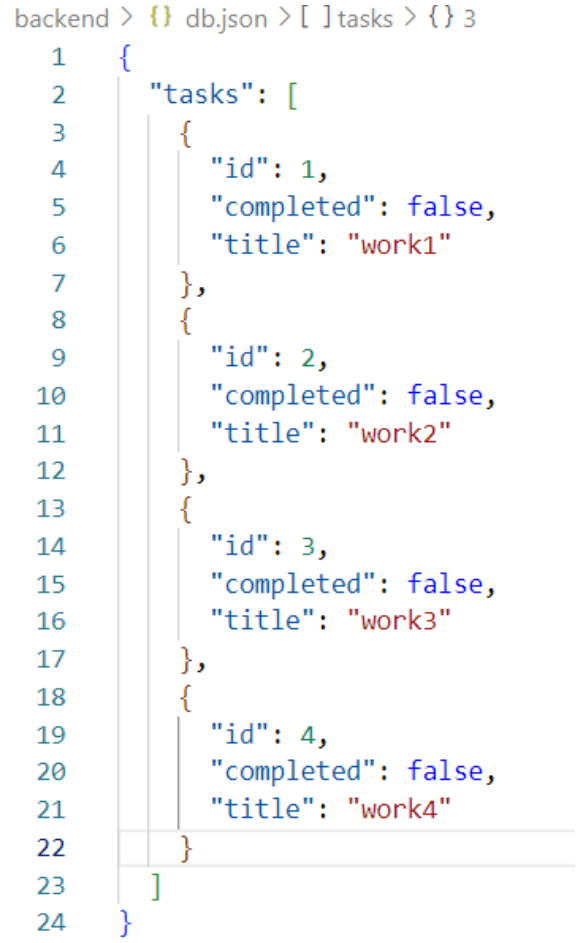
Input Name	Submit Result
register("firstName")	{firstName: 'value'}
register("name.firstName")	{name: { firstName: 'value' }}
register("name.firstName.0")	{name: { firstName: ['value'] }}

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.

5. **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

6. Todo page

a. Call api in TodoApp component

```
12    const { tasks, dispatch } = useContext(TaskContext);
13    const [isPending, setIsPending] = useState(true)
14
15    useEffect(() => {
16      setTimeout(()=>
17      fetch("http://localhost:3001/tasks")
18      .then((rawData)=> rawData.json())
19      .then((data)=>{
20        dispatch({ type: 'INITIALIZE_TASKS', payload: data });
21        setIsPending(false);
22      }), 2000)
23
24    }, [dispatch]);
```

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

```
const navigate = useNavigate();

const logout = ()=>{
  localStorage.isLogin = false;
  navigate('/');
}
```

```
31
32  return (
33    <div className='container'>
34      {isPending && <div>Loading...</div>}
35      {!isPending && tasks && (
36      <div>
37        <p>{localStorage.getItem("username")}</p>
38      <FilterTaskFrom />
39
40      <AddTaskForm />
41
42      <TaskList isCompleted={false}/>
43
44      {tasks.some((item)=> item.completed) > 0 ? (
45        <Accordion defaultActiveKey='0' className='mt-4'>
46          <Accordion.Item eventKey='0'>
47            <Accordion.Header>Completed</Accordion.Header>
48            <Accordion.Body>
49              <TaskList isCompleted={true} />
50            </Accordion.Body>
51          </Accordion.Item>
52        </Accordion>
53      ) : null}
54      </div>)}
55      <Button className="mt-3"onClick={logout}>
56        Logout
57      </Button>
58
59    </div>
```

b. Call Api in add task

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

```
function AddTaskForm() {
  const [newTask, setNewTask] = useState('');
  const [isPending, setIsPending] = useState(false);
  const { dispatch } = useContext(TaskContext);

  const handleSubmit = (e) => {
    e.preventDefault();

    if (newTask.trim() !== '') {
      const task = {
        title: newTask,
        completed: false
      }
      setIsPending(true);

      setTimeout(()=>
    fetch(`http://localhost:3001/tasks`,{
      method: 'POST',
      headers: {"Content-Type": "application/json"},
      body: JSON.stringify(task)
    })
    .then((response)=>{
      if(!response.ok)
        throw Error('Error');
      return response.json();
    })
    .then((data)=>{
      dispatch({
        type: 'ADD_TASK',
        payload: data,
      });
      setNewTask('');
      setIsPending(false);
    }), 2000);
    }
  };
```

And change the way the button display when pending

```
42      }), 2000);
43      }
44    };
45
46    return (
47      <Form onSubmit={handleSubmit} className='d-flex flex-column gap-2'>
48        <Form.Group controlId='addTask'>
49          <Form.Control
50            type='text'
51            placeholder='Add a task'
52            size='sm'
53            value={newTask}
54            onChange={(e) => setNewTask(e.target.value)}
55          />
56        </Form.Group>
57        <Button
58          variant='primary'
59          size='sm'
60          type='submit'
61          className='align-self-end'
62          disabled={isPending}
63        >
64          Add
65          {isPending? <AiOutlineLoading3Quarters /> :<IoMdAddCircleOutline />}
66        </Button>
67      </Form>
68    );
69  }
70
```

c. 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.

```
8   function TaskItem({ task }) {
9     const { dispatch } = useContext(TaskContext);
10    const [isPending, setIsPending] = useState(false);
11
12    const handleToggleComplete = () => {
13      task.completed = !task.completed
14      setIsPending(true);
15
16      setTimeout(()=>
17      fetch(`http://localhost:3001/tasks/${task.id}`,{
18        method: 'PUT',
19        headers: {"Content-Type": "application/json"},
20        body: JSON.stringify(task)
21      })
22      .then((response)=>{
23        if(!response.ok)
24          throw Error('Error');
25        return response.json();
26      })
27      .then((data)=>{
28        dispatch({
29          type: 'TOGGLE_TASK',
30          payload: data,
31        });
32        setIsPending(false);
33      }), 2000);
34    };
35
36    return (
37      <ListGroup.Item variant={task.completed ? 'success' : 'primary'}>
38        <Button
39          variant={task.completed ? 'success' : 'outline-success'}
40          onClick={handleToggleComplete}
41          className="mr-2"
42          disabled={isPending}
43        >
44          {isPending && <AiOutlineLoading3Quarters/> }
45          {!isPending && (task.completed ? <IoMdRemove /> : <IoMdAdd />)}
46        </Button>
47        <span>{task.completed ? <del>{task.title}</del> : task.title}</span>
48      </ListGroup.Item>
49    );
50  }
```

d. Change the reducer logic from the previous version

```
5   const taskReducer = (state, action) => {
6     console.log({ state, action });
7     switch (action.type) {
8       case 'ADD_TASK': {
9         const newTask = [
10          ...state, action.payload
11        ];
12        return newTask;
13      }
14
15      case 'TOGGLE_TASK': {
16        const newTasks = state.map((task) =>
17          task.id === action.payload.id
18            ? action.payload
19            : task
20        );
21        return newTasks;
22      }
23
24      case 'INITIALIZE_TASKS': {
25        return [...action.payload];
26      }
27
28      default:
29        return state;
30    }
31  };
```