PUBLISHED: OCTOBER 09 2020
LAST UPDATED: NOVEMBER 22 2021

React - CRUD Example with React Hook Form

Example built with **React 16.13.1** and **React Hook Form 6.9.2**

Other versions available:

- **React:** Formik (/post/2020/04/17/react-formik-master-details-crud-example)
- **Angular:** Angular 14 (/post/2022/12/21/angular-14-crud-example-with-reactive-forms), 11 (/post/2020/12/15/angular-11-crud-example-with-reactive-forms), 10 (/post/2020/09/01/angular-master-details-crud-example)
- **Next.js:** Next.js 10 (/post/2021/04/20/next-js-10-crud-example-with-react-hook-form)

This tutorial shows how to build a basic React CRUD application with the React Hook Form library that includes pages for listing, adding, editing and deleting records from a JSON API. The records in the example app are user records, but the same CRUD pattern and code structure could be used to manage any type of data e.g. products, services, articles etc.

Fake backend API with CRUD routes

The example app runs with a fake backend api by default to enable it to run completely in the browser without a real api (backend-less), the fake api contains routes for user CRUD operations (Create, Read, Update, Delete) and it uses browser local storage to save data. To disable the fake backend you just have to remove a couple of lines of code from the root index.jsx file, you can build your own api or hook it up with the .NET CRUD api available (instructions below).

React Hook Form Library

React Hook Form is a relatively new library for working with forms in React using React Hooks, I just stumbled across it recently and will be using it for my React projects going forward, I think it's easier to use than the other options available and requires less code.

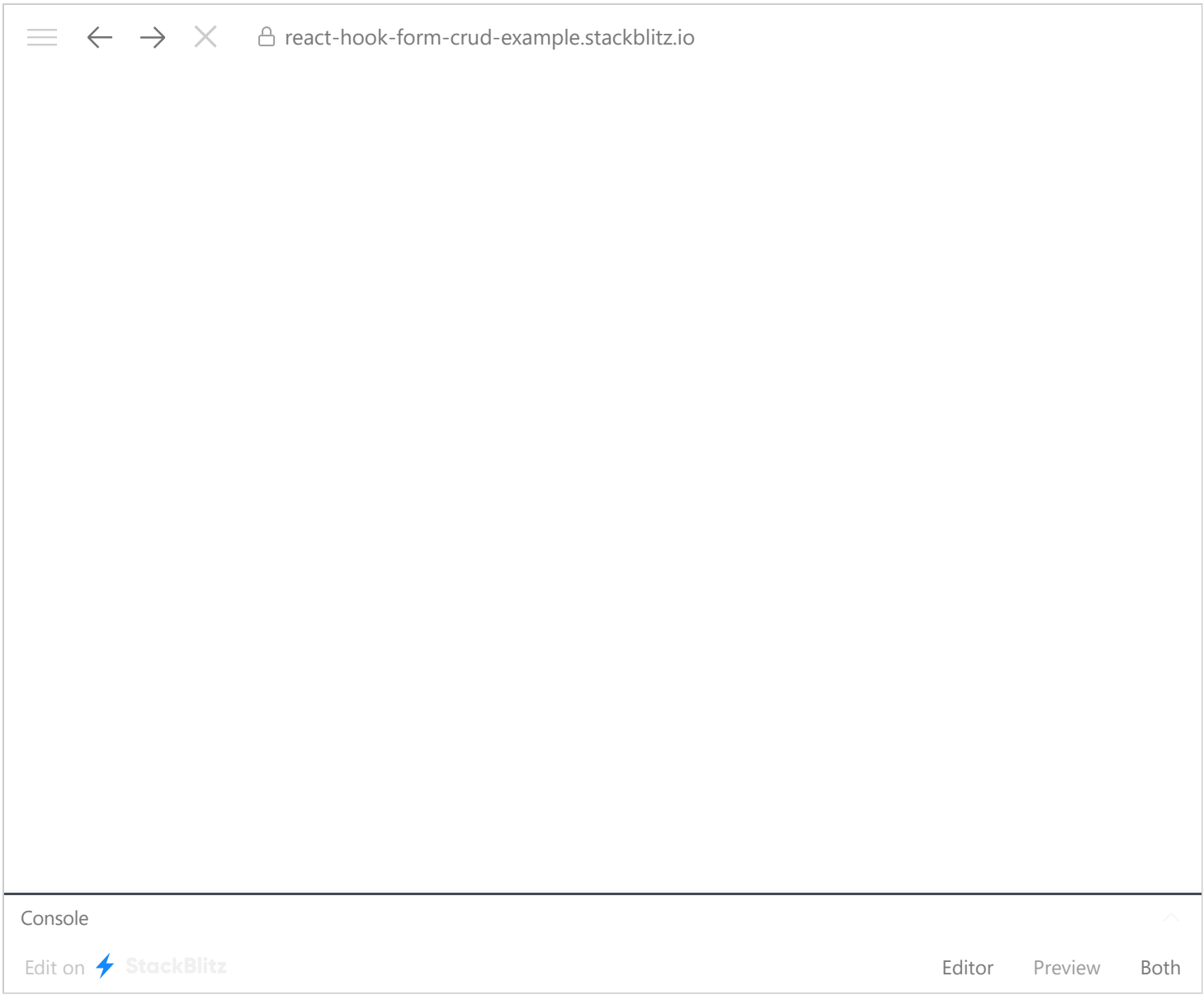
For more info see https://react-hook-form.com (https://react-hook-form.com/).

Example CRUD App Overview

The example app includes a basic home page and users section with CRUD functionality, the default page in the users section displays a list of all users and includes buttons to add, edit and delete users. The add and edit buttons navigate to a page containing a React Hook Form for creating or updating a user record, and the delete button executes a function within the user list component to delete the user record. The add and edit forms are both implemented with the same add/edit component which behaves differently depending on which mode it is in ("add mode" vs "edit mode").

The example project is available on GitHub at https://github.com/cornflourblue/react-hook-form-crud-example (https://github.com/cornflourblue/react-hook-form-crud-example).

Here it is in action:



(See on StackBlitz at https://stackblitz.com/edit/react-hook-form-crud-example (https://stackblitz.com/edit/react-hook-form-crud-example))

Update History:

- 22 Nov 2021 - Uploaded compatible Node.js + SQL API
- 22 Nov 2021 - Updated from Webpack 4 to 5 to fix npm audit warnings
- 28 Sept 2021 - Added compatible CRUD API built with .NET 5.0
- 09 Oct 2020 - Built tutorial with **React 16.13.1** and **React Hook Form 6.9.2**

Run the React CRUD Example Locally

1. Install Node and npm from https://nodejs.org (https://nodejs.org)

2. Download or clone the project source code from https://github.com/cornflourblue/react-hook-form-crud-example (https://github.com/cornflourblue/react-hook-form-crud-example)
3. Install all required npm packages by running `npm install` from the command line in the project root folder (where the package.json is located).
4. Start the application by running `npm start` from the command line in the project root folder, this will launch a browser displaying the application.

For more info on setting up your local React dev environment see React - Setup Development Environment (/post/2020/06/02/react-setup-development-environment).

Run the React CRUD App with a Node.js + MySQL API

For full details about the example Node.js + MySQL API see the tutorial Node.js + MySQL - CRUD API Example and Tutorial (/post/2021/11/22/nodejs-mysql-crud-api-example-and-tutorial). But to get up and running quickly just follow the below steps.

1. Install MySQL Community Server from https://dev.mysql.com/downloads/mysql/ (https://dev.mysql.com/downloads/mysql/) and ensure it is started. Installation instructions are available at https://dev.mysql.com/doc/refman/8.0/en/installing.html (https://dev.mysql.com/doc/refman/8.0/en/installing.html).
2. Download or clone the project source code from https://github.com/cornflourblue/node-mysql-crud-api (https://github.com/cornflourblue/node-mysql-crud-api)
3. Install all required npm packages by running `npm install` or `npm i` from the command line in the project root folder (where the package.json is located).
4. Update the database credentials in `/config.json` to connect to your MySQL server instance.
5. Start the api by running `npm start` from the command line in the project root folder, you should see the message `Server listening on port 4000`.
6. Back in the React example app, remove or comment out the 2 lines below the comment `// setup fake backend` located in the `/src/index.jsx` file, then start the React app and it should now be hooked up with the Node + MySQL API.

Run the React CRUD App with a .NET API

For full details about the example .NET API see the tutorial .NET 5.0 - CRUD API Example and Tutorial (/post/2021/09/28/net-5-crud-api-example-and-tutorial). But to get up and running quickly just follow the below steps.

1. Install the .NET SDK from https://dotnet.microsoft.com/download (https://dotnet.microsoft.com/download).
2. Download or clone the project source code from https://github.com/cornflourblue/dotnet-5-crud-api (https://github.com/cornflourblue/dotnet-5-crud-api)
3. Start the api by running `dotnet run` from the command line in the project root folder (where the WebApi.csproj file is located), you should see the message `Now listening on: http://localhost:4000`.
4. Back in the React example app, remove or comment out the 2 lines below the comment `// setup fake backend` located in the `/src/index.jsx` file, then start the React app and it should now be hooked up with the .NET API.

Project Structure

All source code for the React CRUD app is located in the `/src` folder. Inside the src folder there is a folder per feature (app, home, users) as well as folders for non-feature code that can be shared across different parts of the app (_components, _helpers, _services).

I prefixed non-feature folders with an underscore `_` to group them together and make it easy to distinguish between features and non-features, it also keeps the project folder structure shallow so it's quick to see everything at a glance from the top level and to navigate around the project.

The `index.js` files in each non-feature folder re-export all of the exports from the folder so they can be imported using only the folder path instead of the full path to each file, and to enable importing multiple modules in a single import (e.g. `import { userService, alertService } from '@/_services'`).

I named the root component file in each feature folder `Index.jsx` so it can be imported using only the folder path (e.g. `import { App } from './app';`), removing the need for an extra `index.js` file that re-exports the component.

Click any of the below links to jump down to a description of each file along with it's code:

- src
  - _components
    - Alert.jsx
    - Nav.jsx
    - index.js
  - _helpers
    - fake-backend.js
    - fetch-wrapper.js
    - role.js
    - index.js
  - _services
    - alert.service.js
    - user.service.js
    - index.js
  - app
    - Index.jsx
  - home
    - Index.jsx
  - users
    - AddEdit.jsx
    - Index.jsx
    - List.jsx
  - index.html
  - index.jsx
  - styles.less
- .babelrc
- package.json
- webpack.config.js

Alert Component

Path: /src/_components/Alert.jsx

The alert component controls the adding & removing of bootstrap alerts in the UI, it maintains an array of `alerts` that are rendered in the template returned by the React Hooks function component.

The `useEffect()` hook is used to subscribe to the observable returned from the `alertService.onAlert()` method, this enables the alert component to be notified whenever an alert message is sent to the alert service and add it to the `alerts` array for display. Sending an alert with an empty message to the alert service tells the alert component to clear the alerts array. The `useEffect()` hook is also used to register a route change listener by calling `history.listen()` which automatically clears alerts on route changes.

The empty dependency array `[]` passed as a second parameter to the `useEffect()` hook causes the react hook to only run once when the component mounts, similar to the `componentDidMount()` method in a traditional react class component. The function returned from the `useEffect()` hook cleans up the subscribtions when the component unmounts, similar to the `componentWillUnmount()` method in a traditional react class component.

The `removeAlert()` function removes the specified `alert` object from the array, it allows individual alerts to be closed in the UI.

The `cssClasses()` function returns a corresponding bootstrap alert class for each of the alert types, if you're using something other than bootstrap you could change the CSS classes returned to suit your application.

The returned JSX template renders a bootstrap alert message for each alert in the alerts array.

For more info see React Hooks + Bootstrap - Alert Notifications (/post/2020/04/11/react-hooks-bootstrap-alert-notifications).

```
import React, { useState, useEffect } from 'react';
import { useHistory } from 'react-router-dom';
import PropTypes from 'prop-types';

import { alertService, AlertType } from '../_services';

const propTypes = {
    id: PropTypes.string,
    fade: PropTypes.bool
};

const defaultProps = {
    id: 'default-alert',
    fade: true
};

function Alert({ id, fade }) {
    const history = useHistory();
    const [alerts, setAlerts] = useState([]);

    useEffect(() => {
        // subscribe to new alert notifications
        const subscription = alertService.onAlert(id)
            .subscribe(alert => {
                // clear alerts when an empty alert is received
                if (!alert.message) {
                    setAlerts(alerts => {
                        // filter out alerts without 'keepAfterRouteChange' flag
                        const filteredAlerts = alerts.filter(x => x.keepAfterRouteChange

                        // remove 'keepAfterRouteChange' flag on the rest
                        filteredAlerts.forEach(x => delete x.keepAfterRouteChange);
                        return filteredAlerts;
                    });
                } else {
                    // add alert to array
                    setAlerts(alerts => ([...alerts, alert]));

                    // auto close alert if required
                    if (alert.autoClose) {
                        setTimeout(() => removeAlert(alert), 3000);
                    }
                }
            });
```

```
        // clear alerts on location change
        const historyUnlisten = history.listen(({ pathname }) => {
            // don't clear if pathname has trailing slash because this will be auto redi
            if (pathname.endsWith('/')) return;

            alertService.clear(id);
        });

        // clean up function that runs when the component unmounts
        return () => {
            // unsubscribe & unlisten to avoid memory leaks
            subscription.unsubscribe();
            historyUnlisten();
        };
    }, []);

    function removeAlert(alert) {
        if (fade) {
            // fade out alert
            const alertWithFade = { ...alert, fade: true };
            setAlerts(alerts => alerts.map(x => x === alert ? alertWithFade : x));

            // remove alert after faded out
            setTimeout(() => {
                setAlerts(alerts => alerts.filter(x => x !== alertWithFade));
            }, 250);
        } else {
            // remove alert
            setAlerts(alerts => alerts.filter(x => x !== alert));
        }
    }

    function cssClasses(alert) {
        if (!alert) return;

        const classes = ['alert', 'alert-dismissable'];

        const alertTypeClass = {
            [AlertType.Success]: 'alert alert-success',
            [AlertType.Error]: 'alert alert-danger',
            [AlertType.Info]: 'alert alert-info',
            [AlertType.Warning]: 'alert alert-warning'
        }

        classes.push(alertTypeClass[alert.type]);
```

```
        if (alert.fade) {
            classes.push('fade');
        }

        return classes.join(' ');
    }

    if (!alerts.length) return null;

    return (
        <div className="container">
            <div className="m-3">
                {alerts.map((alert, index) =>
                    <div key={index} className={cssClasses(alert)}>
                        <a className="close" onClick={() => removeAlert(alert)}>&times;
                        <span dangerouslySetInnerHTML={{__html: alert.message}}></span>
                    </div>
                )}
            </div>
        </div>
    );
}

Alert.propTypes = propTypes;
Alert.defaultProps = defaultProps;
export { Alert };
```

Back to top

Nav Component

Path: /src/_components/Nav.jsx

The nav component displays the main navigation in the example. The react router `NavLink` component automatically adds the `active` class to the active nav item so it is highlighted in the UI.

```
import React from 'react';
import { NavLink } from 'react-router-dom';

function Nav() {
    return (
        <nav className="navbar navbar-expand navbar-dark bg-dark">
            <div className="navbar-nav">
                <NavLink exact to="/" className="nav-item nav-link">Home</NavLink>
                <NavLink to="/users" className="nav-item nav-link">Users</NavLink>
            </div>
        </nav>
    );
}

export { Nav };
```

Back to top

Fake Backend

Path: /src/_helpers/fake-backend.js

The fake backend is enabled by executing the below `configureFakeBackend()` function which monkey patches `fetch()` to create a custom fetch function.

The new custom fetch function returns a javascript `Promise` that resolves after a half second delay to simulate a real api call. The fake backend is organised into a top level `handleRoute()` function that checks the request url and method to determine how the request should be handled. For intercepted routes one of the below `// route functions` is called, for all other routes the request is passed through to the real backend via the `realFetch()` function which points to the original `window.fetch` function. Below the route functions there are a few `// helper functions` for returning different response types and performing small tasks.

For more info see React + Fetch - Fake Backend Example for Backendless Development (/post/2020/03/10/react-fetch-fake-backend-example-for-backendless-development).

```
import { Role } from './'

export function configureFakeBackend() {
    // array in local storage for user records
    let users = JSON.parse(localStorage.getItem('users')) || [{
        id: 1,
        title: 'Mr',
        firstName: 'Joe',
        lastName: 'Bloggs',
        email: 'joe@bloggs.com',
        role: Role.User,
        password: 'joe123'
    }];

    // monkey patch fetch to setup fake backend
    let realFetch = window.fetch;
    window.fetch = function (url, opts) {
        return new Promise((resolve, reject) => {
            // wrap in timeout to simulate server api call
            setTimeout(handleRoute, 500);

            function handleRoute() {
                const { method } = opts;
                switch (true) {
                    case url.endsWith('/users') && method === 'GET':
                        return getUsers();
                    case url.match(/\/users\/\d+$/) && method === 'GET':
                        return getUserById();
                    case url.endsWith('/users') && method === 'POST':
                        return createUser();
                    case url.match(/\/users\/\d+$/) && method === 'PUT':
                        return updateUser();
                    case url.match(/\/users\/\d+$/) && method === 'DELETE':
                        return deleteUser();
                    default:
                        // pass through any requests not handled above
                        return realFetch(url, opts)
                            .then(response => resolve(response))
                            .catch(error => reject(error));
                }
            }

            // route functions

            function getUsers() {
```

```
            return ok(users);
        }

        function getUserById() {
            let user = users.find(x => x.id === idFromUrl());
            return ok(user);
        }

        function createUser() {
            const user = body();

            if (users.find(x => x.email === user.email)) {
                return error(`User with the email ${user.email} already exists`);
            }

            // assign user id and a few other properties then save
            user.id = newUserId();
            user.dateCreated = new Date().toISOString();
            delete user.confirmPassword;
            users.push(user);
            localStorage.setItem('users', JSON.stringify(users));

            return ok();
        }

        function updateUser() {
            let params = body();
            let user = users.find(x => x.id === idFromUrl());

            // only update password if included
            if (!params.password) {
                delete params.password;
            }
            // don't save confirm password
            delete params.confirmPassword;

            // update and save user
            Object.assign(user, params);
            localStorage.setItem('users', JSON.stringify(users));

            return ok();
        }

        function deleteUser() {
            users = users.filter(x => x.id !== idFromUrl());
            localStorage.setItem('users', JSON.stringify(users));
```

```
93
94                  return ok();
95              }
96
97              // helper functions
98
99              function ok(body) {
100                 resolve({ ok: true, text: () => Promise.resolve(JSON.stringify(body)) }
101             }
102
103             function error(message) {
104                 resolve({ status: 400, text: () => Promise.resolve(JSON.stringify({ mes
105             }
106
107             function idFromUrl() {
108                 const urlParts = url.split('/');
109                 return parseInt(urlParts[urlParts.length - 1]);
110             }
111
112             function body() {
113                 return opts.body && JSON.parse(opts.body);
114             }
115
116             function newUserId() {
117                 return users.length ? Math.max(...users.map(x => x.id)) + 1 : 1;
118             }
119         });
120     }
};
```

Back to top

Fetch Wrapper

Path: /src/_helpers/fetch-wrapper.js

The fetch wrapper is a lightweight wrapper around the native browser `fetch()` function used to simplify the code for making HTTP requests. It contains methods for `get`, `post`, `put` and `delete` requests, it automatically handles the parsing of JSON data from responses, and throws an error if the HTTP response is not successful (`!response.ok`).

With the fetch wrapper a `POST` request can be made as simply as this: `fetchWrapper.post(url, body);` . It is used in the example app by the user service.

```
export const fetchWrapper = {
    get,
    post,
    put,
    delete: _delete
};

function get(url) {
    const requestOptions = {
        method: 'GET'
    };
    return fetch(url, requestOptions).then(handleResponse);
}

function post(url, body) {
    const requestOptions = {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(body)
    };
    return fetch(url, requestOptions).then(handleResponse);
}

function put(url, body) {
    const requestOptions = {
        method: 'PUT',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(body)
    };
    return fetch(url, requestOptions).then(handleResponse);
}

// prefixed with underscored because delete is a reserved word in javascript
function _delete(url) {
    const requestOptions = {
        method: 'DELETE'
    };
    return fetch(url, requestOptions).then(handleResponse);
}

// helper functions

function handleResponse(response) {
    return response.text().then(text => {
        const data = text && JSON.parse(text);
```

```
        if (!response.ok) {
            const error = (data && data.message) || response.statusText;
            return Promise.reject(error);
        }

        return data;
    });
}
```

Back to top

Role Object / Enum

Path: /src/_helpers/role.js

The role object defines the all the roles in the example application, I created it to use like an enum (https://en.wikipedia.org/wiki/Enumerated_type) to avoid passing roles around as strings, so instead of `'Admin'` we can use `Role.Admin`.

```
export const Role = {
    Admin: 'Admin',
    User: 'User'
};
```

Back to top

Alert Service

Path: /src/_services/alert.service.js

The alert service acts as the bridge between any component in an React application and the alert component that actually displays the alert notification. It contains methods for sending, clearing and subscribing to alerts.

The `AlertType` object defines the types of alerts allowed in the application.

You can trigger alert notifications from any component in the application by calling one of the convenience methods for displaying different types of alerts: `success()`, `error()`, `info()` and `warn()`.

Alert convenience method parameters

- The first parameter is the alert `message` string, which can be plain text or HTML
- The second parameter is an optional `options` object that supports an `autoClose` boolean property and `keepAfterRouteChange` boolean property:
  - `autoClose` - if true tells the alert component to automatically close the alert after three seconds. Default is `true`.
  - `keepAfterRouteChange` - if true prevents the alert from being closed after one route change, this is handy for displaying messages after a redirect such as when a new user is created or updated. Default is `false`.

For more info see React Hooks + Bootstrap - Alert Notifications (/post/2020/04/11/react-hooks-bootstrap-alert-notifications).

```
import { Subject } from 'rxjs';
import { filter } from 'rxjs/operators';

const alertSubject = new Subject();
const defaultId = 'default-alert';

export const alertService = {
    onAlert,
    success,
    error,
    info,
    warn,
    alert,
    clear
};

export const AlertType = {
    Success: 'Success',
    Error: 'Error',
    Info: 'Info',
    Warning: 'Warning'
};

// enable subscribing to alerts observable
function onAlert(id = defaultId) {
    return alertSubject.asObservable().pipe(filter(x => x && x.id === id));
}

// convenience methods
function success(message, options) {
    alert({ ...options, type: AlertType.Success, message });
}

function error(message, options) {
    alert({ ...options, type: AlertType.Error, message });
}

function info(message, options) {
    alert({ ...options, type: AlertType.Info, message });
}

function warn(message, options) {
    alert({ ...options, type: AlertType.Warning, message });
}
```

```
// core alert method
function alert(alert) {
    alert.id = alert.id || defaultId;
    alert.autoClose = (alert.autoClose === undefined ? true : alert.autoClose);
    alertSubject.next(alert);
}

// clear alerts
function clear(id = defaultId) {
    alertSubject.next({ id });
}
```

Back to top

User Service

Path: /src/_services/user.service.js

The user service handles communication between the react app and the backend api, it contains standard CRUD methods for managing users that make corresponding HTTP requests to the `/users` endpoint of the api by calling the fetch wrapper.

```
import config from 'config';
import { fetchWrapper } from '@/_helpers';

const baseUrl = `${config.apiUrl}/users`;

export const userService = {
    getAll,
    getById,
    create,
    update,
    delete: _delete
};

function getAll() {
    return fetchWrapper.get(baseUrl);
}

function getById(id) {
    return fetchWrapper.get(`${baseUrl}/${id}`);
}

function create(params) {
    return fetchWrapper.post(baseUrl, params);
}

function update(id, params) {
    return fetchWrapper.put(`${baseUrl}/${id}`, params);
}

// prefixed with underscored because delete is a reserved word in javascript
function _delete(id) {
    return fetchWrapper.delete(`${baseUrl}/${id}`);
}
```

Back to top

App Index Component

Path: /src/app/Index.jsx

The `App` component is the root component of the example app, it contains the outer html, main nav, global alert, and top level routes for the application.

As a convention I named the root component file in each feature folder `Index.jsx` so it can be imported using only the folder path (`import { App } from './app';`), removing the need for an extra `index.js` file that re-exports the App component.

The first route (<Redirect from="/:url*(/+)" to={pathname.slice(0, -1)} />) automatically removes trailing slashes from URLs which can cause issues and are a side-effect of using relative react router links. For more info see React Router - Relative Links Example (/post/2020/03/26/react-router-relative-links-example).

The last route (<Redirect from="*" to="/" />) is a catch-all redirect route that redirects any unmatched paths to the home page.

```
import React from 'react';
import { Route, Switch, Redirect, useLocation } from 'react-router-dom';

import { Nav, Alert } from '@/_components';
import { Home } from '@/home';
import { Users } from '@/users';

function App() {
    const { pathname } = useLocation();

    return (
        <div className="app-container bg-light">
            <Nav />
            <Alert />
            <div className="container pt-4 pb-4">
                <Switch>
                    <Redirect from="/:url*(/+)" to={pathname.slice(0, -1)} />
                    <Route exact path="/" component={Home} />
                    <Route path="/users" component={Users} />
                    <Redirect from="*" to="/" />
                </Switch>
            </div>
        </div>
    );
}

export { App };
```

Back to top

Home Index Component

Path: /src/home/Index.jsx

The `Home` component is a basic react function component that displays some HTML and a link to the users page.

As a convention I named the root component file in each feature folder `Index.jsx` so it can be imported using only the folder path (`import { Home } from '@/home';`), removing the need for an extra `index.js` file that re-exports the Home component.

```
import React from 'react';
import { Link } from 'react-router-dom';

function Home() {
    return (
        <div>
            <h1>React - CRUD Example with React Hook Form</h1>
            <p>An example app showing how to list, add, edit and delete user records wi
            <p><Link to="users">&gt;&gt; Manage Users</Link></p>
        </div>
    );
}

export { Home };
```

Back to top

Users Add/Edit Component

Path: /src/users/AddEdit.jsx

The users `AddEdit` component is used for both adding and editing users, it contains a form built with the React Hook Form library.

Form validation rules are defined with the Yup schema validation library and passed to the React Hook Form `useForm()` function, for more info on Yup see https://github.com/jquense/yup (https://github.com/jquense/yup).

The `useForm()` hook function returns an object with methods for working with a form including registering inputs, handling form submit, resetting the form, setting input values, displaying errors and more, for a complete list see https://react-hook-form.com/api#useForm (https://react-hook-form.com/api#useForm).

The `onSubmit` function gets called when the form is submitted and valid, and either creates or updates a user depending on which mode it is in.

The form is in "add mode" when there is no user id parameter (`match.params.id`), otherwise it is in "edit mode". The variable `isAddMode` is used to change the form behaviour based on the mode it is in, for example in "add mode" the password field is required, and in "edit mode" (`!isAddMode`) the user service is called to get the user details and set the field values.

The returned JSX template contains the form with all of the input fields and validation messages. The form fields are registered with the React Hook Form using the `ref={register}` attribute which registers each input with the input `name` . For more info on form validation with React Hook Form see React - Form Validation Example with React Hook Form (/post/2020/10/04/react-form-validation-example-with-react-hook-form).

```
import React, { useEffect } from 'react';
import { Link } from 'react-router-dom';
import { useForm } from "react-hook-form";
import { yupResolver } from '@hookform/resolvers/yup';
import * as Yup from 'yup';

import { userService, alertService } from '@/_services';

function AddEdit({ history, match }) {
    const { id } = match.params;
    const isAddMode = !id;

    // form validation rules
    const validationSchema = Yup.object().shape({
        title: Yup.string()
            .required('Title is required'),
        firstName: Yup.string()
            .required('First Name is required'),
        lastName: Yup.string()
            .required('Last Name is required'),
        email: Yup.string()
            .email('Email is invalid')
            .required('Email is required'),
        role: Yup.string()
            .required('Role is required'),
        password: Yup.string()
            .transform(x => x === '' ? undefined : x)
            .concat(isAddMode ? Yup.string().required('Password is required') : null)
            .min(6, 'Password must be at least 6 characters'),
        confirmPassword: Yup.string()
            .transform(x => x === '' ? undefined : x)
            .when('password', (password, schema) => {
                if (password || isAddMode) return schema.required('Confirm Password is r
            })
            .oneOf([Yup.ref('password')], 'Passwords must match')
    });

    // functions to build form returned by useForm() hook
    const { register, handleSubmit, reset, setValue, errors, formState } = useForm({
        resolver: yupResolver(validationSchema)
    });

    function onSubmit(data) {
        return isAddMode
            ? createUser(data)
```

```
            : updateUser(id, data);
    }

    function createUser(data) {
        return userService.create(data)
            .then(() => {
                alertService.success('User added', { keepAfterRouteChange: true });
                history.push('.');
            })
            .catch(alertService.error);
    }

    function updateUser(id, data) {
        return userService.update(id, data)
            .then(() => {
                alertService.success('User updated', { keepAfterRouteChange: true });
                history.push('..');
            })
            .catch(alertService.error);
    }

    useEffect(() => {
        if (!isAddMode) {
            // get user and set form fields
            userService.getById(id).then(user => {
                const fields = ['title', 'firstName', 'lastName', 'email', 'role'];
                fields.forEach(field => setValue(field, user[field]));
            });
        }
    }, []);

    return (
        <form onSubmit={handleSubmit(onSubmit)} onReset={reset}>
            <h1>{isAddMode ? 'Add User' : 'Edit User'}</h1>
            <div className="form-row">
                <div className="form-group col">
                    <label>Title</label>
                    <select name="title" ref={register} className={`form-control ${erro
                        <option value=""></option>
                        <option value="Mr">Mr</option>
                        <option value="Mrs">Mrs</option>
                        <option value="Miss">Miss</option>
                        <option value="Ms">Ms</option>
                    </select>
                    <div className="invalid-feedback">{errors.title?.message}</div>
                </div>
```

```
                <div className="form-group col-5">
                    <label>First Name</label>
                    <input name="firstName" type="text" ref={register} className={`form
                    <div className="invalid-feedback">{errors.firstName?.message}</div>
                </div>
                <div className="form-group col-5">
                    <label>Last Name</label>
                    <input name="lastName" type="text" ref={register} className={`form-
                    <div className="invalid-feedback">{errors.lastName?.message}</div>
                </div>
            </div>
            <div className="form-row">
                <div className="form-group col-7">
                    <label>Email</label>
                    <input name="email" type="text" ref={register} className={`form-con
                    <div className="invalid-feedback">{errors.email?.message}</div>
                </div>
                <div className="form-group col">
                    <label>Role</label>
                    <select name="role" ref={register} className={`form-control ${error
                        <option value=""></option>
                        <option value="User">User</option>
                        <option value="Admin">Admin</option>
                    </select>
                    <div className="invalid-feedback">{errors.role?.message}</div>
                </div>
            </div>
            {!isAddMode &&
                <div>
                    <h3 className="pt-3">Change Password</h3>
                    <p>Leave blank to keep the same password</p>
                </div>
            }
            <div className="form-row">
                <div className="form-group col">
                    <label>Password</label>
                    <input name="password" type="password" ref={register} className={`fo
                    <div className="invalid-feedback">{errors.password?.message}</div>
                </div>
                <div className="form-group col">
                    <label>Confirm Password</label>
                    <input name="confirmPassword" type="password" ref={register} classNa
                    <div className="invalid-feedback">{errors.confirmPassword?.message}<
                </div>
            </div>
            <div className="form-group">
```

```
                <button type="submit" disabled={formState.isSubmitting} className="btn l
                    {formState.isSubmitting && <span className="spinner-border spinner-l
                    Save
                </button>
                <Link to={isAddMode ? '.' : '..'} className="btn btn-link">Cancel</Link
            </div>
        </form>
    );
}

export { AddEdit };
```

Back to top

Users Index Component

Path: /src/users/Index.jsx

The `Users` component is the root component of the users section / feature, it defines routes for each of the pages within the users section.

The first route matches the root users path (`/users`) making it the default route for this section, so by default the users `List` component is displayed. The second and third routes are for adding and editing users, they match different routes but both load the same component (`AddEdit`), the component modifies its behaviour based on the route.

As a convention I named the root component file in each feature folder `Index.jsx` so it can be imported using only the folder path (`import { Users } from '@/users';`), removing the need for an extra `index.js` file that re-exports the Users component.

```
import React from 'react';
import { Route, Switch } from 'react-router-dom';

import { List } from './List';
import { AddEdit } from './AddEdit';

function Users({ match }) {
    const { path } = match;

    return (
        <Switch>
            <Route exact path={path} component={List} />
            <Route path={`${path}/add`} component={AddEdit} />
            <Route path={`${path}/edit/:id`} component={AddEdit} />
        </Switch>
    );
}

export { Users };
```

Back to top

Users List Component

Path: /src/users/List.jsx

The users `List` component displays a list of all users and contains buttons for adding, editing and deleting users. A `useEffect` hook is used to get all users from the user service and store them in local state by calling `setUsers()`.

The delete button calls the `deleteUser()` function which first updates the user is local state with an `isDeleting = true` property so the UI displays a spinner on the delete button, it then calls `userService.delete()` to delete the user and then removes the deleted user from local state so it is removed from the UI.

```
import React, { useState, useEffect } from 'react';
import { Link } from 'react-router-dom';

import { userService } from '@/_services';

function List({ match }) {
    const { path } = match;
    const [users, setUsers] = useState(null);

    useEffect(() => {
        userService.getAll().then(x => setUsers(x));
    }, []);

    function deleteUser(id) {
        setUsers(users.map(x => {
            if (x.id === id) { x.isDeleting = true; }
            return x;
        }));
        userService.delete(id).then(() => {
            setUsers(users => users.filter(x => x.id !== id));
        });
    }

    return (
        <div>
            <h1>Users</h1>
            <Link to={`${path}/add`} className="btn btn-sm btn-success mb-2">Add User</
            <table className="table table-striped">
                <thead>
                    <tr>
                        <th style={{ width: '30%' }}>Name</th>
                        <th style={{ width: '30%' }}>Email</th>
                        <th style={{ width: '30%' }}>Role</th>
                        <th style={{ width: '10%' }}></th>
                    </tr>
                </thead>
                <tbody>
                    {users && users.map(user =>
                        <tr key={user.id}>
                            <td>{user.title} {user.firstName} {user.lastName}</td>
                            <td>{user.email}</td>
                            <td>{user.role}</td>
                            <td style={{ whiteSpace: 'nowrap' }}>
                                <Link to={`${path}/edit/${user.id}`} className="btn btn
                                <button onClick={() => deleteUser(user.id)} className="
```

```
                                {user.isDeleting
                                    ? <span className="spinner-border spinner-borde
                                    : <span>Delete</span>
                                }
                            </button>
                        </td>
                    </tr>
                )}
                {!users &&
                    <tr>
                        <td colSpan="4" className="text-center">
                            <div className="spinner-border spinner-border-lg align-
                        </td>
                    </tr>
                }
                {users && !users.length &&
                    <tr>
                        <td colSpan="4" className="text-center">
                            <div className="p-2">No Users To Display</div>
                        </td>
                    </tr>
                }
            </tbody>
        </table>
    </div>
  );
}

export { List };
```

Back to top

Base Index HTML File

Path: /src/index.html

The base index html file contains the outer html for the whole react CRUD example application. When the app is started with `npm start`, webpack bundles up all of the react code into a single javascript file and injects it into the body of this page to be loaded by the browser. The react app is then rendered in the `<div id="app"></div>` element by the main react entry file below.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <base href="/" />
    <title>React - CRUD Example with React Hook Form</title>

    <!-- bootstrap css -->
    <link href="//netdna.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="s
</head>
<body>
    <div id="app"></div>
</body>
</html>
```

Back to top

Main React Entry File

Path: /src/index.jsx

The root index.jsx file bootstraps the React Hook Form CRUD application by rendering the `App` component (wrapped in a react router `BrowserRouter`) into the `app` div element defined in the base index html file above.

The boilerplate application uses a fake / mock backend that stores data in browser local storage, to disable the fake backend simply remove the 2 lines of code below the comment `// setup fake backend`.

```
import React from 'react';
import { BrowserRouter } from 'react-router-dom';
import { render } from 'react-dom';

import { App } from './App';

import './styles.less';

// setup fake backend
import { configureFakeBackend } from './_helpers';
configureFakeBackend();

render(
    <BrowserRouter>
        <App />
    </BrowserRouter>,
    document.getElementById('app')
);
```

Back to top

Global LESS/CSS Styles

Path: /src/styles.less

The styles.less file contains global custom LESS/CSS styles for the example react app. For more info see React - How to add Global CSS / LESS styles to React with webpack (/post/2020/02/10/react-how-to-add-global-css-less-styles-to-react-with-webpack).

```
// global styles
a { cursor: pointer; }

.app-container {
    min-height: 350px;
}

.btn-delete-user {
    width: 40px;
    text-align: center;
    box-sizing: content-box;
}
```

Back to top

Babel RC (Run Commands) File

Path: /.babelrc

The babel config file defines the presets used by babel to transpile the React and ES6 code. The babel transpiler is run by webpack via the `babel-loader` module configured in the webpack.config.js file below.

```
{
    "presets": [
        "@babel/preset-react",
        "@babel/preset-env"
    ]
}
```

Back to top

Package.json

Path: /package.json

The package.json file contains project configuration information including package dependencies which get installed when you run `npm install`. Full documentation is available on the npm docs website (https://docs.npmjs.com/files/package.json).

```
{
    "name": "react-hook-form-crud-example",
    "version": "1.0.0",
    "repository": {
        "type": "git",
        "url": "https://github.com/cornflourblue/react-hook-form-crud-example.git"
    },
    "license": "MIT",
    "scripts": {
        "build": "webpack --mode production",
        "start": "webpack-dev-server --open"
    },
    "dependencies": {
        "@hookform/resolvers": "^1.0.0",
        "prop-types": "^15.7.2",
        "react": "^16.13.1",
        "react-dom": "^16.13.1",
        "react-hook-form": "^6.9.2",
        "react-router-dom": "^5.1.2",
        "rxjs": "^6.5.5",
        "yup": "^0.29.3"
    },
    "devDependencies": {
        "@babel/core": "^7.4.3",
        "@babel/preset-env": "^7.4.3",
        "@babel/preset-react": "^7.0.0",
        "babel-loader": "^8.0.5",
        "css-loader": "^4.3.0",
        "html-webpack-plugin": "^4.5.0",
        "less": "^3.11.0",
        "less-loader": "^7.0.1",
        "path": "^0.12.7",
        "style-loader": "^1.1.3",
        "webpack": "^5.64.2",
        "webpack-cli": "^4.9.1",
        "webpack-dev-server": "^4.5.0"
    }
}
```

Back to top

Webpack Config

Path: /webpack.config.js

Webpack is used to compile and bundle all the project files so they're ready to be loaded into a browser, it does this with the help of loaders and plugins that are configured in the webpack.config.js file. For more info about webpack check out the webpack docs (https://webpack.js.org/concepts/).

The webpack config file also defines a global config object for the React CRUD application using the `externals` property, you can also use this to define different config variables for your development and production environments.

```
var HtmlWebpackPlugin = require('html-webpack-plugin');
const path = require('path');

module.exports = {
    mode: 'development',
    module: {
        rules: [
            {
                test: /\.jsx?$/,
                loader: 'babel-loader'
            },
            {
                test: /\.less$/,
                use: [
                    { loader: 'style-loader' },
                    { loader: 'css-loader' },
                    { loader: 'less-loader' }
                ]
            }
        ]
    },
    resolve: {
        mainFiles: ['index', 'Index'],
        extensions: ['.js', '.jsx'],
        alias: {
            '@': path.resolve(__dirname, 'src/'),
        }
    },
    plugins: [new HtmlWebpackPlugin({
        template: './src/index.html'
    })],
    devServer: {
        historyApiFallback: true
    },
    externals: {
        // global app config object
        config: JSON.stringify({
            apiUrl: 'http://localhost:4000'
        })
    }
}
```

Back to top

Need Some React Help?

Search fiverr (https://fvrr.co/396SRIm) to find help quickly from experienced React freelance developers.

Follow me for updates

I share all new blog posts on Twitter (https://twitter.com/jason_watmore) and Facebook (https://www.facebook.com/JasonWatmoreBlog).

When I'm not coding

My wife and I are attempting to ride motorcycles around Australia and vlog about it on YouTube (https://www.youtube.com/TinaAndJason?sub_confirmation=1).

More React Posts

- Next.js 13 + App Router + MongoDB - User Rego and Login Tutorial with Example (/next-js-13-app-router-mongodb-user-rego-and-login-tutorial-with-example)
- Next.js 13 - Fix for client component ('use client') hangs when fetching data in useEffect hook (/next-js-13-fix-for-client-component-use-client-hangs-when-fetching-data-in-useeffect-hook)
- Add Google AdSense to a Single Page App - React, Angular, Vue, Next etc... (/add-google-adsense-to-a-single-page-app-react-angular-vue-next-etc)
- Next.js 13 + MySQL - User Registration and Login Tutorial with Example App (/next-js-13-mysql-user-registration-and-login-tutorial-with-example-app)
- Next.js Router - Listen to route (location) change with useRouter (/nextjs-router-listen-to-route-location-change-with-userouter)
- Next.js 13 + MongoDB - User Registration and Login Tutorial with Example App (/next-js-13-mongodb-user-registration-and-login-tutorial-with-example-app)
- React Router v6 - Redirect with Navigate and useNavigate (/react-router-v6-redirect-with-navigate-and-usenavigate)
- Redux Toolkit createAsyncThunk - Dispatch a Redux Action from an Async Thunk in React with RTK (/redux-toolkit-createasyncthunk-dispatch-a-redux-action-from-

an-async-thunk-in-react-with-rtk)
- React 18 + Redux - User Registration and Login Example & Tutorial (/react-18-redux-user-registration-and-login-example-tutorial)
- React Router v6 - Catch All (Default) Redirect in React (/react-router-v6-catch-all-default-redirect-in-react)

Show more...

Comments

ABOUT

I'm a web developer in Sydney Australia and co-founder of Point Blank Development (https://www.pointblankdevelopment.com.au), I've been coding since 1998.

Other than coding I'm attempting a big lap of Australia by motorcycle with my wife Tina, you can follow the adventure on YouTube (https://www.youtube.com/TinaAndJason?sub_confirmation=1) & TinaAndJason.com.au (https://tinaandjason.com.au).

Find me on: (https://twitter.com/jason_watmore)
(https://www.facebook.com/JasonWatmoreBlog)
(https://github.com/cornflourblue)
(https://www.youtube.com/c/JasonWatmore)

Subscribe to feed: RSS (/rss), Atom (/atom), JSON (/json)

You can support the blog on Patreon (https://www.patreon.com/jasonwatmore)

Code Snippets (/page/code-snippets)

MONTHS

2023
- August (/posts/2023/08) (2)
- July (/posts/2023/07) (3)
- June (/posts/2023/06) (4)
- May (/posts/2023/05) (6)
- April (/posts/2023/04) (19)
- March (/posts/2023/03) (11)
- February (/posts/2023/02) (45)
- January (/posts/2023/01) (17)

2022
2021
2020

2019
2018
2017
2016
2015
2014
2013
2012
2011

TAGS

Alerts, Angular, Angular 10, Angular 11, Angular 14, Angular 15, Angular 16, Angular 2, Angular 4, Angular 5, Angular 6, Angular 7, Angular 8, Angular 9, Angular Directive, Angular UI Router, AngularJS, Animation, ASP.NET, ASP.NET Core, ASP.NET Web API, Authentication and Authorization, AWS, Axios, Azure, Basic Authentication, Blazor, Bootstrap, C#, Chai, CKEditor, CSS3, Dapper, DDD, Deployment, Design Patterns, Docker, Dynamic LINQ, EF Core, ELMAH, ES6, Exchange, Facebook, Fetch, Fluent NHibernate, Formik, Git, GitHub, Google, Google Analytics, Google API, Google Maps API, Google Plus, Heroku, HTML5, HTTP, IIS, Insecure Content, Instagram API, Ionic Framework, iOS, iPhone, Java, JavaScript, jQuery, JWT, LinkedIn, LINQ, Login, MEAN Stack, MERN Stack, MEVN Stack, Mocha, Modal, MongoDB, Moq, MSSQL, MVC, MVC5, MySQL, .NET, NextJS, NGINX, ngMock, NHibernate, Ninject, NodeJS, npm, Pagination, Pinia, Pinterest, PostgreSQL, Razor Pages, React, React Hook Form, React Hooks, Recoil, Redmine, Redux, Registration, Repository, RxJS, Security, Sequelize, Shell Scripting, Sinon, SinonJS, SQLite, SSH, TDD, Terraform, Twitter, TypeScript, Ubuntu, Umbraco, Unit of Work, Unit Testing, URL Rewrite, Validation, Vanilla JS, VS Code, Vue, Vue 3, Vuex, Webpack, Windows, Windows Server 2008, Zustand (/posts/tag/zustand)