

React Redux CRUD example with API calls

 bezkoder.com/react-redux-crud-example/

In this tutorial, I will show you how to build a React Redux CRUD Application example to consume Rest API, display and modify data with Router, Axios & Bootstrap.

Related Posts:

- React File Upload with Axios and Progress Bar to Rest API
- React Redux: JWT Authentication example
- React JWT Authentication (without Redux) example

Serverless:

- React Firebase CRUD with Realtime Database
- React Firestore CRUD App example | Firebase Cloud Firestore

Using Hooks instead:

React Hooks + Redux: CRUD example with Axios and Rest API

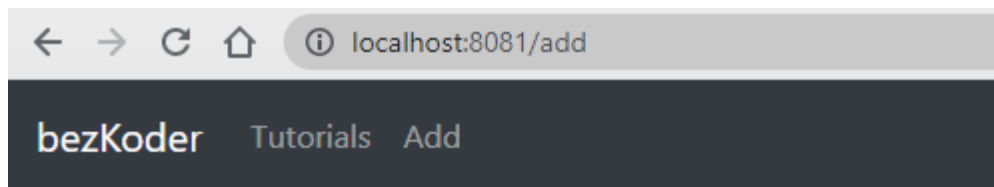
Overview of React Redux CRUD example with Rest API

We will build a React Redux Tutorial Application with API calls in that:

- Each Tutorial has id, title, description, published status.
- We can create, retrieve, update, delete Tutorials.
- There is a Search bar for finding Tutorials by title.

Here are screenshots of our React Redux CRUD Application.

- Create an item:



Title

React Redux CRUD example

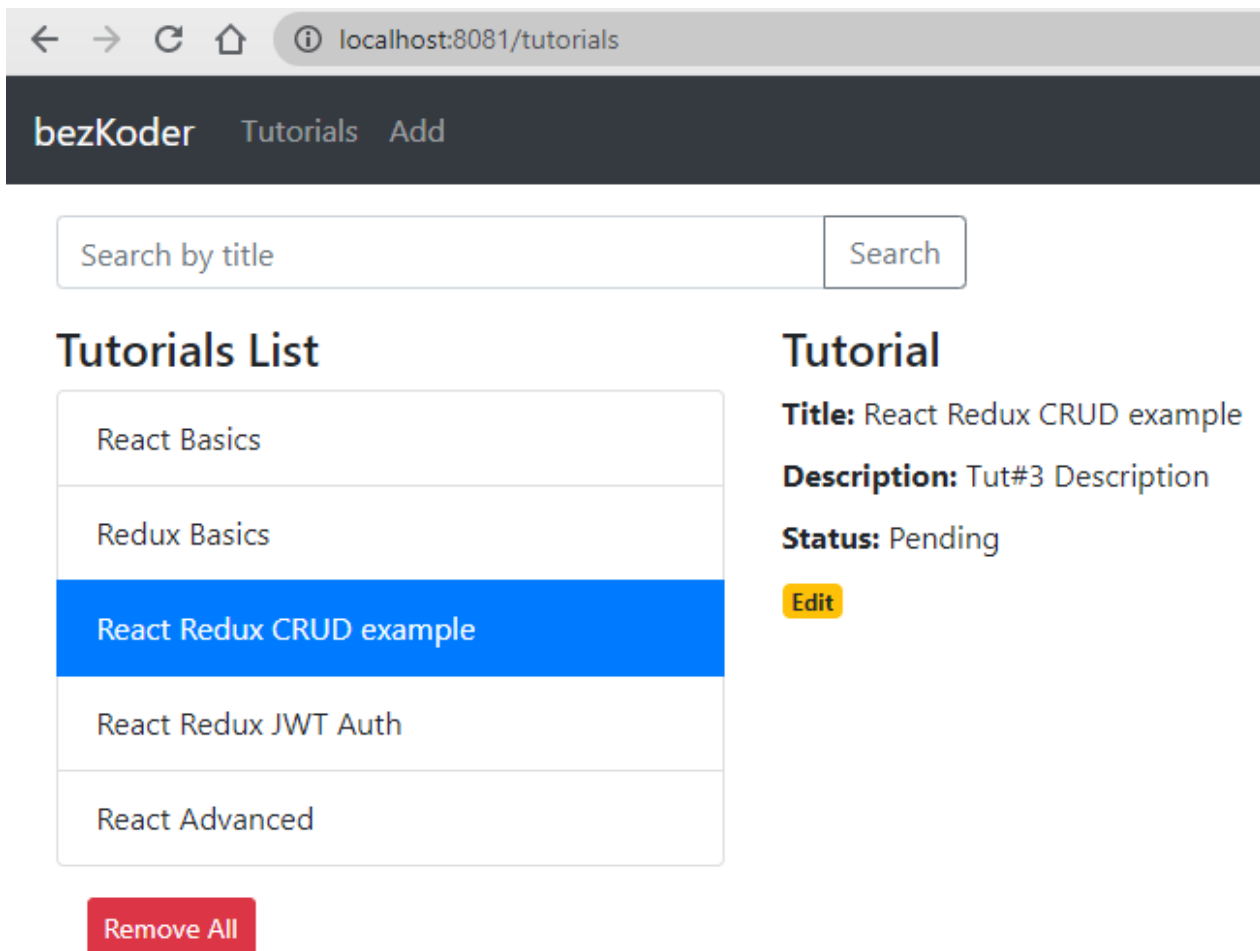
Description

Tut#3 Description

Submit

If you want to implement Form Validation, please visit:
React Form Validation example

– Retrieve all items:



– Click on **Edit** button to update an item:

← → ↻ 🏠 ⓘ localhost:8081/tutorials/3

bezKoder Tutorials Add

Tutorial

Title

React Redux CRUD example

Description

Tut#3 Description

Status:Pending

Publish

Delete

Update

On this Page, you can:

- change status to **Published** using **Publish** button
- delete the item using **Delete** button
- update the item details with **Update** button

← → ↻ 🏠 ⓘ localhost:8081/tutorials/3

bezKoder Tutorials Add

Tutorial

Title

React Redux CRUD example (updated)

Description

Tut#3 Description (new)

Status:Published

UnPublish

Delete

Update

The tutorial was updated successfully!

– Search Tutorials by title:

ux

Search

Tutorials List

Redux Basics

React Redux CRUD example (updated)

React Redux JWT Auth

Remove All

Tutorial

Title:

React Redux CRUD example (updated)

Description:

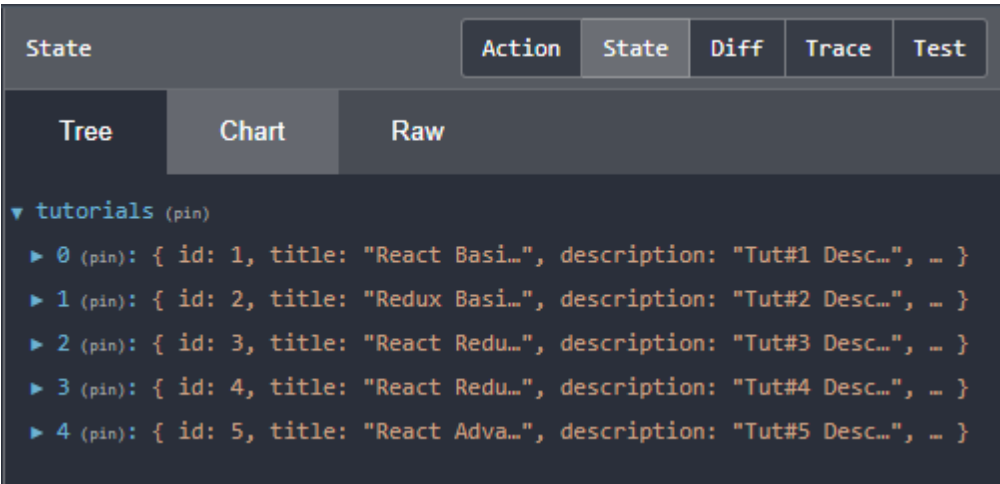
Tut#3 Description (new)

Status:

Published

Edit

– Redux Store:



This React Client consumes the following Web API:

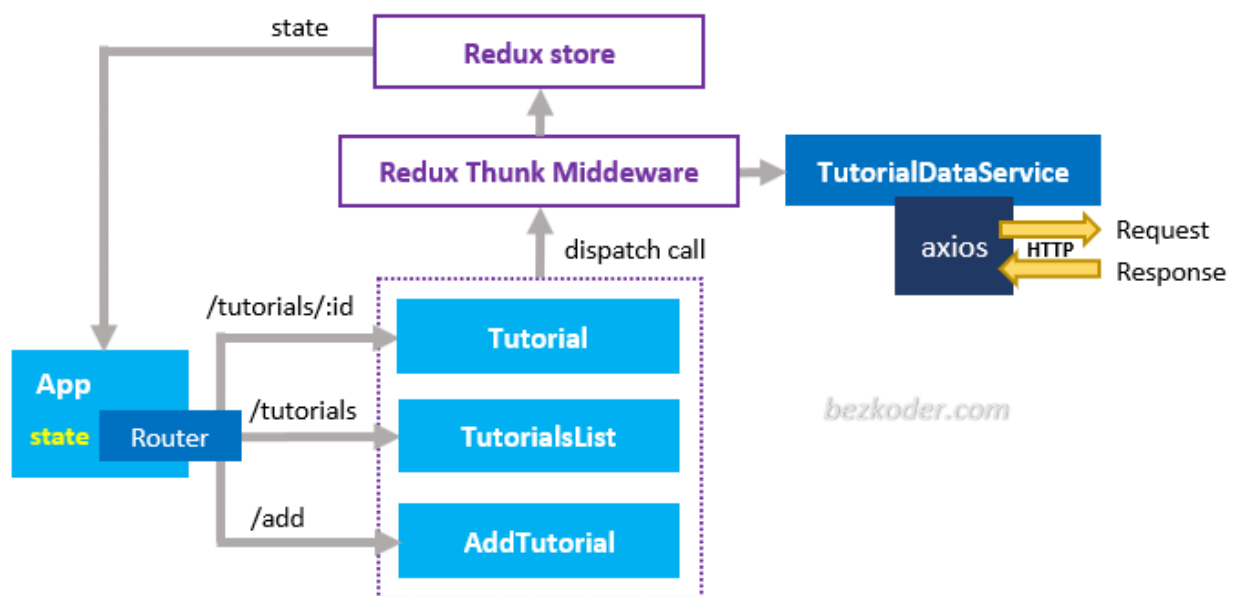
Methods	Urls	Actions
POST	/api/tutorials	create new Tutorial
GET	/api/tutorials	retrieve all Tutorials
GET	/api/tutorials/:id	retrieve a Tutorial by :id
PUT	/api/tutorials/:id	update a Tutorial by :id
DELETE	/api/tutorials/:id	delete a Tutorial by :id
DELETE	/api/tutorials	delete all Tutorials
GET	/api/tutorials?title=[keyword]	find all Tutorials which title contains keyword

You can find step by step to build a Server like this in one of these posts:

- Express, Sequelize & MySQL
- Express, Sequelize & PostgreSQL
- Express, Sequelize & SQL Server
- Express & MongoDB
- Spring Boot & MySQL
- Spring Boot & PostgreSQL
- Spring Boot & MongoDB
- Spring Boot & SQL Server
- Spring Boot & H2
- Spring Boot & Cassandra
- Spring Boot & Oracle
- Python/Django & MySQL
- Python/Django & PostgreSQL
- Python/Django & MongoDB

React Redux App Component Diagram with Router & Axios

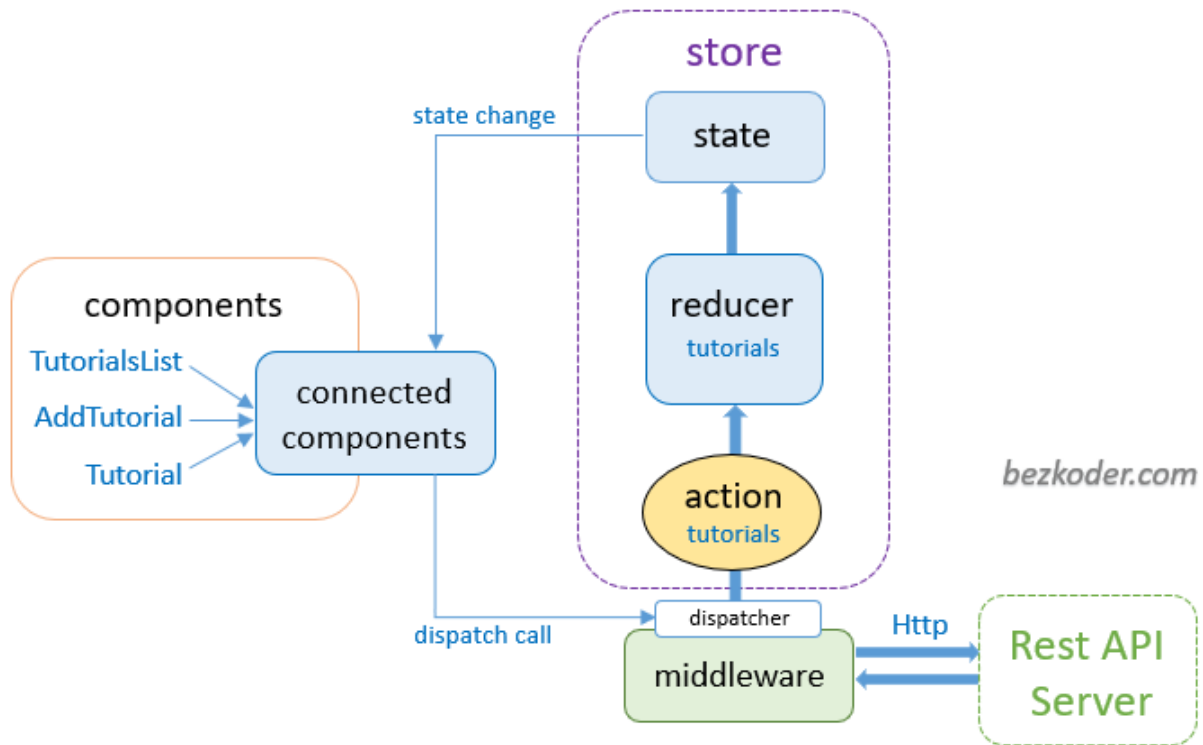
Now look at the React components that we're gonna implement:



- The **App** component is a container with React **Router**. It has **navbar** that links to routes paths.
- Three components that dispatch **actions** to **Redux Thunk Middleware** which uses **TutorialDataService** to call Rest API.
 - **TutorialsList** component gets and displays Tutorials.
 - **Tutorial** component has form for editing Tutorial's details based on **:id**.
 - **AddTutorial** component has form for submission new Tutorial.
- **TutorialDataService** uses **axios** to make HTTP requests and receive responses.

React Redux with API example

This diagram shows how Redux elements work in our React Application:



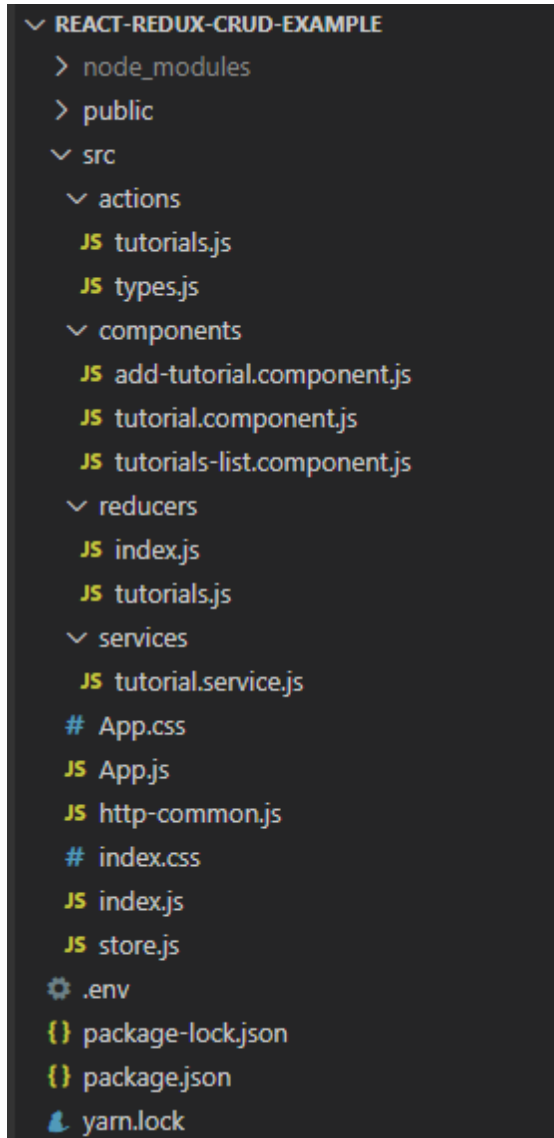
We're gonna create Redux **store** for storing **tutorials** data. Other React Components will work with the Store via dispatching an **action**.

The **reducer** will take the action and return new **state**.

Technology

- React 17/16
- react-redux 7.2.3
- redux 4.0.5
- redux-thunk 2.3.0
- react-router-dom 5.2.0
- axios 0.21.1
- bootstrap 4

Project Structure



I'm gonna explain it briefly.

- **package.json** contains main modules: `react`, `react-router-dom`, `react-redux`, `redux`, `redux-thunk`, `axios` & `bootstrap`.
- `App` is the container that has `Router` & navbar.
- There are 3 components: `TutorialsList`, `Tutorial`, `AddTutorial`.
- **http-common.js** initializes axios with HTTP base Url and headers.
- `TutorialDataService` has methods for sending HTTP requests to the Apis.
- **.env** configures *port* for this React CRUD App.

About Redux elements that we're gonna use:

- **actions** folder contains the action creator (*tutorials.js* for CRUD operations and searching).
- **reducers** folder contains the reducer (*tutorials.js*) which updates the application state corresponding to dispatched action.

If you want to use Redux-Toolkit instead, kindly visit:
Redux-Toolkit example with CRUD Application

Setup React.js Project

Open cmd at the folder you want to save Project folder, run command:

```
npx create-react-app react-redux-crud-example
```

After the process is done. We create additional folders and files like the following tree:

```
public
src
actions
types.js
tutorials.js (create/retrieve/update/delete actions)
reducers
index.js
tutorials.js
components
add-tutorial.component.js
tutorial.component.js
tutorials-list.component.js
services
tutorial.service.js
App.css
App.js
index.js
store.js
package.json
```

Import Bootstrap to React Redux CRUD App

Run command: `npm install bootstrap@4.6.0.`

Open **src/App.js** and modify the code inside it as following-

```
import React, { Component } from "react";
import "bootstrap/dist/css/bootstrap.min.css";

class App extends Component {
  render() {
    // ...
  }
}

export default App;
```

Add React Router to React Redux CRUD App

- Run the command: `npm install react-router-dom`.
- Open **src/App.js** and wrap all UI elements by **BrowserRouter** object.

```
...
import { BrowserRouter as Router } from "react-router-dom";

class App extends Component {
  render() {
    return (
      <Router>
        ...
      </Router>
    );
  }
}

export default App;
```

Add Navbar to React Redux CRUD App

The **App** component is the root container for our application, it will contain a **navbar** inside **<Router>** above, and also, a **Switch** object with several **Route**. Each **Route** points to a React Component.

Now **App.js** looks like:

```

import React, { Component } from "react";
import { BrowserRouter as Router, Switch, Route, Link } from "react-router-dom";
import "bootstrap/dist/css/bootstrap.min.css";
import "./App.css";

import AddTutorial from "./components/add-tutorial.component";
import Tutorial from "./components/tutorial.component";
import TutorialList from "./components/tutorials-list.component";

class App extends Component {
  render() {
    return (
      <Router>
        <nav className="navbar navbar-expand navbar-dark bg-dark">
          <Link to={"/tutorials"} className="navbar-brand">
            bezKoder
          </Link>
          <div className="navbar-nav mr-auto">
            <li className="nav-item">
              <Link to={"/tutorials"} className="nav-link">
                Tutorials
              </Link>
            </li>
            <li className="nav-item">
              <Link to={"/add"} className="nav-link">
                Add
              </Link>
            </li>
          </div>
        </nav>

        <div className="container mt-3">
          <Switch>
            <Route exact path={["/", "/tutorials"]} component={TutorialList} />
            <Route exact path="/add" component={AddTutorial} />
            <Route path="/tutorials/:id" component={Tutorial} />
          </Switch>
        </div>
      </Router>
    );
  }
}

export default App;

```

Initialize Axios for React Redux CRUD with API calls

Let's install *axios* with command: `npm install axios`.

Under **src** folder, we create *http-common.js* file with following code:

```
import axios from "axios";

export default axios.create({
  baseURL: "http://localhost:8080/api",
  headers: {
    "Content-type": "application/json"
  }
});
```

You can change the **baseURL** that depends on REST APIs url that your Server configures.

For more details about ways to use Axios, please visit:

Axios request: Get/Post/Put/Delete example

Create Data Service

In this step, we're gonna create a service that uses axios object above to send HTTP requests or make API calls.

services/*tutorial.service.js*

```
import http from "../http-common";

class TutorialDataService {
  getAll() {
    return http.get("/tutorials");
  }

  get(id) {
    return http.get(`/tutorials/${id}`);
  }

  create(data) {
    return http.post("/tutorials", data);
  }

  update(id, data) {
    return http.put(`/tutorials/${id}`, data);
  }

  delete(id) {
    return http.delete(`/tutorials/${id}`);
  }

  deleteAll() {
    return http.delete(`/tutorials`);
  }

  findByTitle(title) {
    return http.get(`/tutorials?title=${title}`);
  }
}

export default new TutorialDataService();
```

We call axios **get**, **post**, **put**, **delete** method corresponding to HTTP Requests: GET, POST, PUT, DELETE to make CRUD Operations.

You can simplify import statement with:
Absolute Import in React

Create Redux Actions

We're gonna create actions in **src/actions** folder:

actions

types.js

tutorials.js (*create/retrieve/update/delete actions*)

Action Types

First we defined some string constant that indicates the type of action being performed.

actions/type.js

```
export const CREATE_TUTORIAL = "CREATE_TUTORIAL";
export const RETRIEVE_TUTORIALS = "RETRIEVE_TUTORIALS";
export const UPDATE_TUTORIAL = "UPDATE_TUTORIAL";
export const DELETE_TUTORIAL = "DELETE_TUTORIAL";
export const DELETE_ALL_TUTORIALS = "DELETE_ALL_TUTORIALS";
```

Actions Creator

This is creator for actions related to tutorials. We're gonna import `TutorialDataService` to make asynchronous HTTP requests with trigger `dispatch` on the result.

– `createTutorial()`

- calls the `TutorialDataService.create()`
- dispatch `CREATE_TUTORIAL`

– `retrieveTutorials()`

- calls the `TutorialDataService.getAll()`
- dispatch `RETRIEVE_TUTORIALS`

– `updateTutorial()`

- calls the `TutorialDataService.update()`
- dispatch `UPDATE_TUTORIAL`

– `deleteTutorial()`

- calls the `TutorialDataService.delete()`
- dispatch `DELETE_TUTORIAL`

– `deleteAllTutorials()`

- calls the `TutorialDataService.deleteAll()`
- dispatch `DELETE_ALL_TUTORIALS`

– `findTutorialsByTitle()`

- calls the `TutorialDataService.findByTitle()`
- dispatch `RETRIEVE_TUTORIALS`

Some action creators return a `Promise` for Components using them.

actions/tutorials.js

```

import {
  CREATE_TUTORIAL,
  RETRIEVE_TUTORIALS,
  UPDATE_TUTORIAL,
  DELETE_TUTORIAL,
  DELETE_ALL_TUTORIALS
} from "../types";

import TutorialDataService from "../services/tutorial.service";

export const createTutorial = (title, description) => async (dispatch) => {
  try {
    const res = await TutorialDataService.create({ title, description });

    dispatch({
      type: CREATE_TUTORIAL,
      payload: res.data,
    });

    return Promise.resolve(res.data);
  } catch (err) {
    return Promise.reject(err);
  }
};

export const retrieveTutorials = () => async (dispatch) => {
  try {
    const res = await TutorialDataService.getAll();

    dispatch({
      type: RETRIEVE_TUTORIALS,
      payload: res.data,
    });
  } catch (err) {
    console.log(err);
  }
};

export const updateTutorial = (id, data) => async (dispatch) => {
  try {
    const res = await TutorialDataService.update(id, data);

    dispatch({
      type: UPDATE_TUTORIAL,
      payload: data,
    });

    return Promise.resolve(res.data);
  } catch (err) {
    return Promise.reject(err);
  }
};

export const deleteTutorial = (id) => async (dispatch) => {
  try {

```

```

    await TutorialDataService.delete(id);

    dispatch({
      type: DELETE_TUTORIAL,
      payload: { id },
    });
  } catch (err) {
    console.log(err);
  }
};

export const deleteAllTutorials = () => async (dispatch) => {
  try {
    const res = await TutorialDataService.deleteAll();

    dispatch({
      type: DELETE_ALL_TUTORIALS,
      payload: res.data,
    });

    return Promise.resolve(res.data);
  } catch (err) {
    return Promise.reject(err);
  }
};

export const findTutorialsByTitle = (title) => async (dispatch) => {
  try {
    const res = await TutorialDataService.findByTitle(title);

    dispatch({
      type: RETRIEVE_TUTORIALS,
      payload: res.data,
    });
  } catch (err) {
    console.log(err);
  }
};

```

Create Redux Reducer

There will be a reducer in **src/reducers** folder, the reducer updates the state corresponding to dispatched Redux actions.

reducers

index.js

tutorials.js

Tutorials Reducer

The **tutorials** reducer will update **tutorials** state of the Redux store:

reducers/tutorials.js

```
import {
  CREATE_TUTORIAL,
  RETRIEVE_TUTORIALS,
  UPDATE_TUTORIAL,
  DELETE_TUTORIAL,
  DELETE_ALL_TUTORIALS,
} from "../actions/types";

const initialState = [];

function tutorialReducer(tutorials = initialState, action) {
  const { type, payload } = action;

  switch (type) {
    case CREATE_TUTORIAL:
      return [...tutorials, payload];

    case RETRIEVE_TUTORIALS:
      return payload;

    case UPDATE_TUTORIAL:
      return tutorials.map((tutorial) => {
        if (tutorial.id === payload.id) {
          return {
            ...tutorial,
            ...payload,
          };
        } else {
          return tutorial;
        }
      });

    case DELETE_TUTORIAL:
      return tutorials.filter(({ id }) => id !== payload.id);

    case DELETE_ALL_TUTORIALS:
      return [];

    default:
      return tutorials;
  }
};

export default tutorialReducer;
```

Combine Reducers

reducers/index.js


```
import { combineReducers } from "redux";
import tutorials from "../tutorials";

export default combineReducers({
  tutorials,
});
```

Because we only have a single store in a Redux application. We use reducer composition instead of many stores to split data handling logic.

For example, if you have Auth Reducer that manages authentication logic, you can use `combineReducers()` like following code:

```
import { combineReducers } from "redux";
import tutorials from "../tutorials";
import auth from "../auth";

export default combineReducers({
  tutorials,
  auth
});
```

Create Redux Store

This Store will bring Actions and Reducers together and hold the Application state.

Now we need to install *Redux*, *Thunk Middleware* and *Redux Devtool Extension*.

Run the command:

```
npm install redux react-redux redux-thunk
npm install --save-dev redux-devtools-extension
```

In the previous section, we used `combineReducers()` to combine 2 reducers into one. Let's import it, and pass it to `createStore()`:

store.js

```
import { createStore, applyMiddleware } from 'redux';
import { composeWithDevTools } from "redux-devtools-extension";
import thunk from 'redux-thunk';
import rootReducer from './reducers';

const initialState = {};

const middleware = [thunk];

const store = createStore(
  rootReducer,
  initialState,
  composeWithDevTools(applyMiddleware(...middleware))
);

export default store;
```

Provide State to React Components

We will use `mapStateToProps` and `mapDispatchToProps` to connect Redux state to React Components' props later using `connect()` function:

```
export default connect(mapStateToProps, mapDispatchToProps)(ReactComponent);
```

So we need to make the Redux `store` available to the `connect()` call in the Components. We will wrap a parent or ancestor Component in `Provider`.

`Provider` is an high order component that wraps up React application and makes it aware of the entire Redux `store`. That is, it provides the `store` to its child components.

Now we want our entire React App to access the `store`, just put the `App` Component within `Provider`.

Open `src/index.js`

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
...
import { Provider } from 'react-redux';
import store from './store';

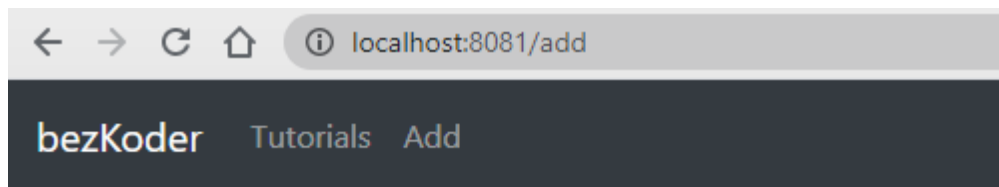
ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById('root')
);
```

Create React Components

Now we're gonna build 3 components corresponding to 3 Routes defined before.

Add item Component

This component has a Form to submit new Tutorial with 2 fields: `title` & `description`.



Title

React Redux CRUD example

Description

Tut#3 Description

Submit

components/add-tutorial.component.js

```

import React, { Component } from "react";
import { connect } from "react-redux";
import { createTutorial } from "../actions/tutorials";

class AddTutorial extends Component {
  constructor(props) {
    super(props);
    this.onChangeTitle = this.onChangeTitle.bind(this);
    this.onChangeDescription = this.onChangeDescription.bind(this);
    this.saveTutorial = this.saveTutorial.bind(this);
    this.newTutorial = this.newTutorial.bind(this);

    this.state = {
      id: null,
      title: "",
      description: "",
      published: false,

      submitted: false,
    };
  }

  onChangeTitle(e) {
    this.setState({
      title: e.target.value,
    });
  }

  onChangeDescription(e) {
    this.setState({
      description: e.target.value,
    });
  }

  saveTutorial() {
    const { title, description } = this.state;

    this.props
      .createTutorial(title, description)
      .then((data) => {
        this.setState({
          id: data.id,
          title: data.title,
          description: data.description,
          published: data.published,

          submitted: true,
        });
        console.log(data);
      })
      .catch((e) => {
        console.log(e);
      });
  }
}

```

```

newTutorial() {
  this.setState({
    id: null,
    title: "",
    description: "",
    published: false,

    submitted: false,
  });
}

render() {
  return (
    ...
  );
}
}

export default connect(null, { createTutorial })(AddTutorial);

```

First, we define the constructor and set initial state, bind `this` to the different events.

Because there are 2 fields, so we create 2 functions to track the values of the input and set that state for changes. We also have a function to get value of the form (state) and send the POST request to the Web API. It dispatch action with `createTutorial()` action creator.

For `render()` method, we check the `submitted` state, if it is true, we show **Add** button for creating new Tutorial again. Otherwise, a Form will display.

```

class AddTutorial extends Component {
  ...

  render() {
    return (
      <div className="submit-form">
        {this.state.submitted ? (
          <div>
            <h4>You submitted successfully!</h4>
            <button className="btn btn-success" onClick={this.newTutorial}>
              Add
            </button>
          </div>
        ) : (
          <div>
            <div className="form-group">
              <label htmlFor="title">Title</label>
              <input
                type="text"
                className="form-control"
                id="title"
                required
                value={this.state.title}
                onChange={this.onChangeTitle}
                name="title"
              />
            </div>

            <div className="form-group">
              <label htmlFor="description">Description</label>
              <input
                type="text"
                className="form-control"
                id="description"
                required
                value={this.state.description}
                onChange={this.onChangeDescription}
                name="description"
              />
            </div>

            <button onClick={this.saveTutorial} className="btn btn-success">
              Submit
            </button>
          </div>
        )}
      </div>
    );
  }
}

```

```

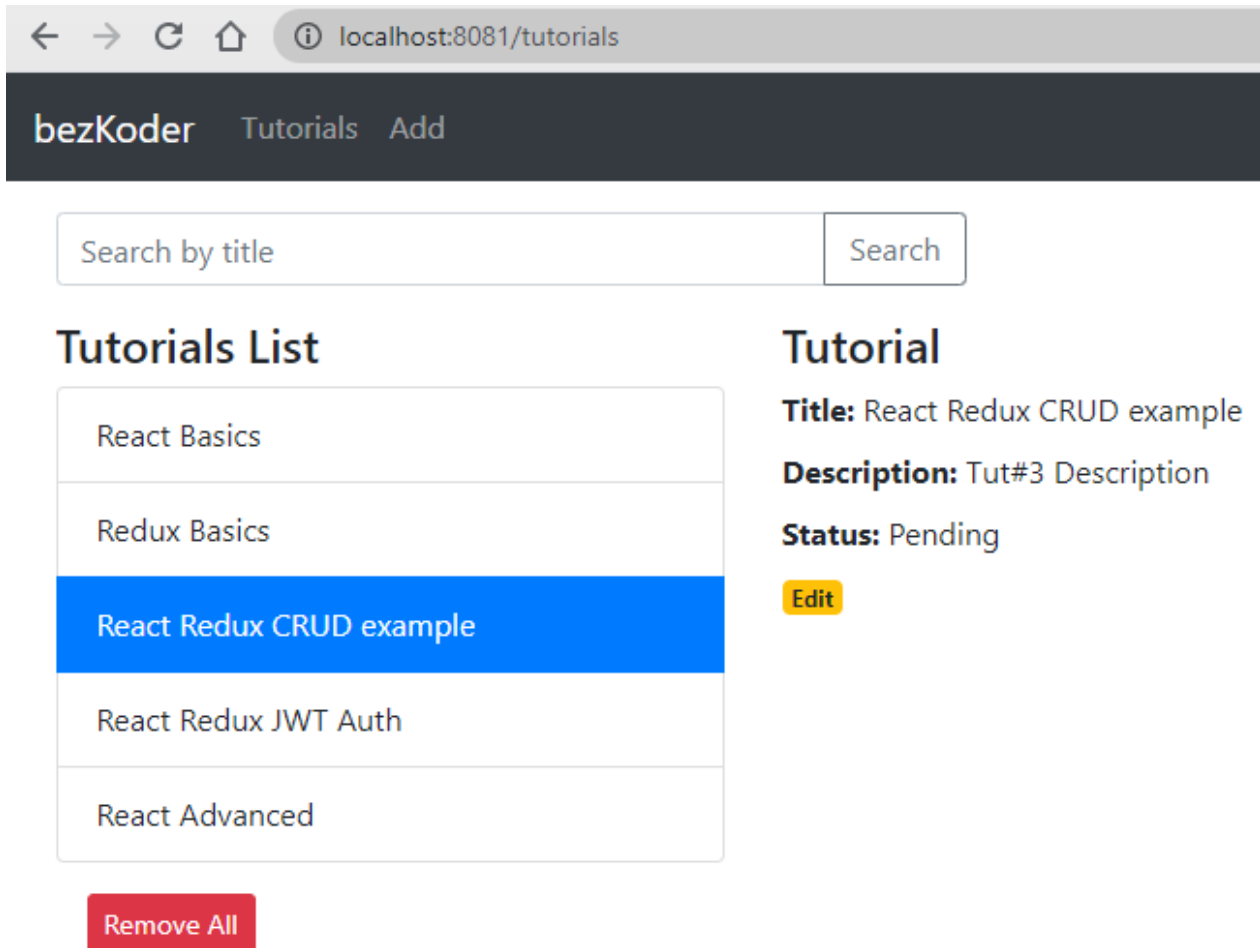
export default connect(null, { createTutorial })(AddTutorial);

```

List of items Component

This component has:

- a search bar for finding Tutorials by *title*.
- a *tutorials* array displayed as a list on the left.
- a selected Tutorial which is shown on the right.



Beside global state `tutorials`, we also have following local state:

- `searchTitle`
- `currentTutorial` and `currentIndex`

We also need to use 3 action creators:

- `retrieveTutorials`
- `findTutorialsByTitle`
- `deleteAllTutorials`

To connect the Redux store with local Component state and props, we use `connect()` with `mapStateToProps` and `mapDispatchToProps`.

```
const mapStateToProps = (state) => {
  return {
    tutorials: state.tutorials
  };
}

export default connect(mapStateToProps, {
  retrieveTutorials,
  findTutorialsByTitle,
  deleteAllTutorials
})(TutorialsList);
```

Now we can work with **tutorials** state and dispatch actions like this:

```
// get tutorials state
this.props.tutorials

// dispatch actions
this.props.retrieveTutorials()
this.props.findTutorialsByTitle(...)
this.props.deleteAllTutorials(...)
```

components/tutorials-list.component.js


```

import React, { Component } from "react";
import { connect } from "react-redux";
import { retrieveTutorials, findTutorialsByTitle, deleteAllTutorials } from
"../actions/tutorials";

class TutorialList extends Component {
  constructor(props) {
    super(props);
    this.onChangeSearchTitle = this.onChangeSearchTitle.bind(this);
    this.refreshData = this.refreshData.bind(this);
    this.setActiveTutorial = this.setActiveTutorial.bind(this);
    this.findByTitle = this.findByTitle.bind(this);
    this.removeAllTutorials = this.removeAllTutorials.bind(this);

    this.state = {
      currentTutorial: null,
      currentIndex: -1,
      searchTitle: "",
    };
  }

  componentDidMount() {
    this.props.retrieveTutorials();
  }

  onChangeSearchTitle(e) {
    const searchTitle = e.target.value;

    this.setState({
      searchTitle: searchTitle,
    });
  }

  refreshData() {
    this.setState({
      currentTutorial: null,
      currentIndex: -1,
    });
  }

  setActiveTutorial(tutorial, index) {
    this.setState({
      currentTutorial: tutorial,
      currentIndex: index,
    });
  }

  removeAllTutorials() {
    this.props
      .deleteAllTutorials()
      .then((response) => {
        console.log(response);
        this.refreshData();
      })
      .catch((e) => {

```

```

        console.log(e);
    });
}

findByTitle() {
    this.refreshData();

    this.props.findTutorialsByTitle(this.state.searchTitle);
}

render() {
    const { searchTitle, currentTutorial, currentIndex } = this.state;
    const { tutorials } = this.props;

    return (
        ...
    );
}

const mapStateToProps = (state) => {
    return {
        tutorials: state.tutorials,
    };
};

export default connect(mapStateToProps, { retrieveTutorials, findTutorialsByTitle,
deleteAllTutorials })(TutorialsList);

```

Let's continue to implement `render()` method:

```

// ...
import { Link } from "react-router-dom";

class TutorialList extends Component {
  ..

  render() {
    const { searchTitle, currentTutorial, currentIndex } = this.state;
    const { tutorials } = this.props;

    return (
      <div className="list row">
        <div className="col-md-8">
          <div className="input-group mb-3">
            <input
              type="text"
              className="form-control"
              placeholder="Search by title"
              value={searchTitle}
              onChange={this.onChangeSearchTitle}
            />
            <div className="input-group-append">
              <button
                className="btn btn-outline-secondary"
                type="button"
                onClick={this.findByTitle}
              >
                Search
              </button>
            </div>
          </div>
        </div>
        <div className="col-md-6">
          <h4>Tutorials List</h4>

          <ul className="list-group">
            {tutorials &&
              tutorials.map((tutorial, index) => (
                <li
                  className={
                    "list-group-item " +
                    (index === currentIndex ? "active" : "")
                  }
                  onClick={() => this.setActiveTutorial(tutorial, index)}
                  key={index}
                >
                  {tutorial.title}
                </li>
              ))}
          </ul>

          <button
            className="m-3 btn btn-sm btn-danger"
            onClick={this.removeAllTutorials}
          >

```

```

        Remove All
      </button>
    </div>
    <div className="col-md-6">
      {currentTutorial ? (
        <div>
          <h4>Tutorial</h4>
          <div>
            <label>
              <strong>Title:</strong>
            </label>{" "}
            {currentTutorial.title}
          </div>
          <div>
            <label>
              <strong>Description:</strong>
            </label>{" "}
            {currentTutorial.description}
          </div>
          <div>
            <label>
              <strong>Status:</strong>
            </label>{" "}
            {currentTutorial.published ? "Published" : "Pending"}
          </div>

          <Link
            to={"/tutorials/" + currentTutorial.id}
            className="badge badge-warning"
          >
            Edit
          </Link>
        </div>
      ) : (
        <div>
          <br />
          <p>Please click on a Tutorial...</p>
        </div>
      )}
    </div>
  </div>
);
}
}

...
export default connect(...)(TutorialsList);

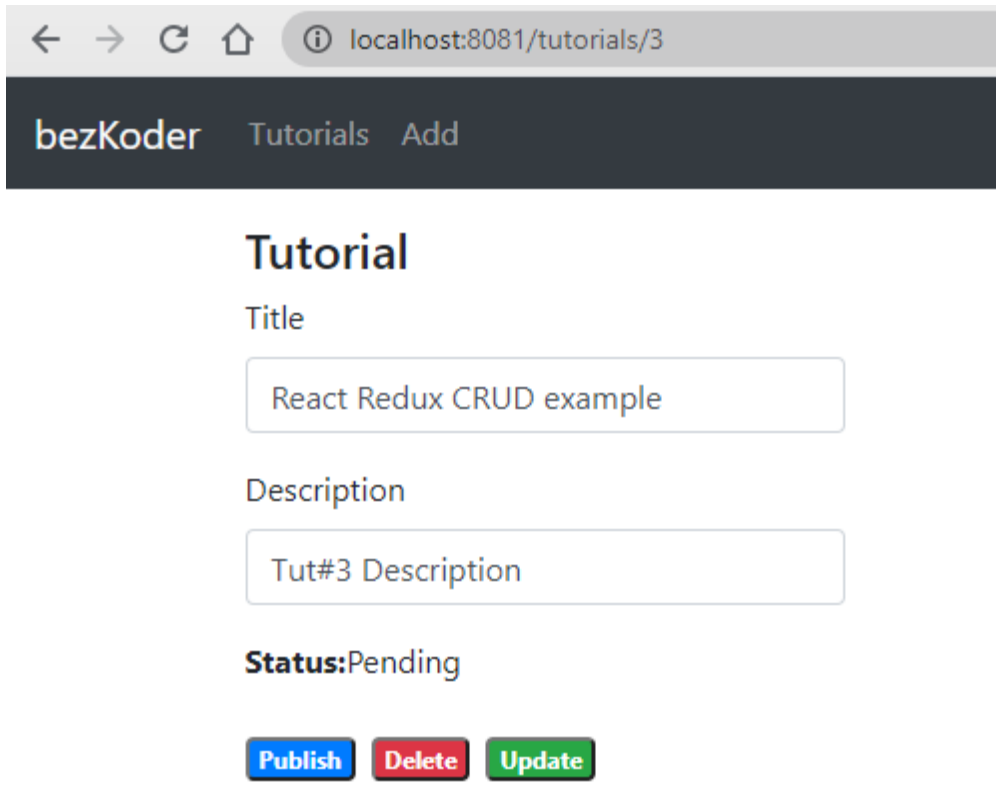
```

If you click on **Edit** button of any Tutorial, the app will direct you to *Tutorial* page. We use React Router **Link** for accessing that page with url: **/tutorials/:id**.

You can add Pagination to this Component, just follow instruction in the post: [React Pagination with API using Material-UI](#)

Item details Component

We're gonna use the component lifecycle method: `componentDidMount()` to fetch the data from the Web API.



The screenshot shows a web browser at `localhost:8081/tutorials/3`. The page has a dark header with the text "bezKoder" and navigation links "Tutorials" and "Add". The main content area is titled "Tutorial" and contains a form with two input fields: "Title" with the value "React Redux CRUD example" and "Description" with the value "Tut#3 Description". Below the form, the status is displayed as "Status: Pending". At the bottom, there are three buttons: "Publish" (blue), "Delete" (red), and "Update" (green).

For getting tutorial details, this component will use `TutorialDataService.get()` method. For update, delete the Tutorial, we work with following action creators:

- `updateTutorial`
- `deleteTutorial`

components*/tutorial.component.js*

```
import React, { Component } from "react";
import { connect } from "react-redux";
import { updateTutorial, deleteTutorial } from "../actions/tutorials";
import TutorialDataService from "../services/tutorial.service";
```

```
class Tutorial extends Component {
  constructor(props) {
    super(props);
    this.onChangeTitle = this.onChangeTitle.bind(this);
    this.onChangeDescription = this.onChangeDescription.bind(this);
    this.getTutorial = this.getTutorial.bind(this);
    this.updateStatus = this.updateStatus.bind(this);
    this.updateContent = this.updateContent.bind(this);
    this.removeTutorial = this.removeTutorial.bind(this);

    this.state = {
      currentTutorial: {
        id: null,
        title: "",
        description: "",
        published: false,
      },
      message: "",
    };
  }

  componentDidMount() {
    this.getTutorial(this.props.match.params.id);
  }

  onChangeTitle(e) {
    const title = e.target.value;

    this.setState(function (prevState) {
      return {
        currentTutorial: {
          ...prevState.currentTutorial,
          title: title,
        },
      };
    });
  }

  onChangeDescription(e) {
    const description = e.target.value;

    this.setState((prevState) => ({
      currentTutorial: {
        ...prevState.currentTutorial,
        description: description,
      },
    }));
  }

  getTutorial(id) {
```

```

TutorialDataService.get(id)
  .then((response) => {
    this.setState({
      currentTutorial: response.data,
    });
    console.log(response.data);
  })
  .catch((e) => {
    console.log(e);
  });
}

updateStatus(status) {
  var data = {
    id: this.state.currentTutorial.id,
    title: this.state.currentTutorial.title,
    description: this.state.currentTutorial.description,
    published: status,
  };

  this.props
    .updateTutorial(this.state.currentTutorial.id, data)
    .then((reponse) => {
      console.log(reponse);

      this.setState((prevState) => ({
        currentTutorial: {
          ...prevState.currentTutorial,
          published: status,
        },
      }));

      this.setState({ message: "The status was updated successfully!" });
    })
    .catch((e) => {
      console.log(e);
    });
}

updateContent() {
  this.props
    .updateTutorial(this.state.currentTutorial.id, this.state.currentTutorial)
    .then((reponse) => {
      console.log(reponse);

      this.setState({ message: "The tutorial was updated successfully!" });
    })
    .catch((e) => {
      console.log(e);
    });
}

removeTutorial() {
  this.props
    .deleteTutorial(this.state.currentTutorial.id)

```

```
        .then(() => {
            this.props.history.push("/tutorials");
        })
        .catch((e) => {
            console.log(e);
        });
    }

    render() {
        const { currentTutorial } = this.state;

        return (
            ...
        );
    }
}

export default connect(null, { updateTutorial, deleteTutorial })(Tutorial);
```

And this is the code for `render()` method:

...

```
class Tutorial extends Component {
  ...

  render() {
    const { currentTutorial } = this.state;

    return (
      <div>
        {currentTutorial ? (
          <div className="edit-form">
            <h4>Tutorial</h4>
            <form>
              <div className="form-group">
                <label htmlFor="title">Title</label>
                <input
                  type="text"
                  className="form-control"
                  id="title"
                  value={currentTutorial.title}
                  onChange={this.onChangeTitle}
                />
              </div>
              <div className="form-group">
                <label htmlFor="description">Description</label>
                <input
                  type="text"
                  className="form-control"
                  id="description"
                  value={currentTutorial.description}
                  onChange={this.onChangeDescription}
                />
              </div>

              <div className="form-group">
                <label>
                  <strong>Status:</strong>
                </label>
                {currentTutorial.published ? "Published" : "Pending"}
              </div>
            </form>

            {currentTutorial.published ? (
              <button
                className="badge badge-primary mr-2"
                onClick={() => this.updateStatus(false)}
              >
                UnPublish
              </button>
            ) : (
              <button
                className="badge badge-primary mr-2"
                onClick={() => this.updateStatus(true)}
              >
```

```

        Publish
      </button>
    )}

    <button
      className="badge badge-danger mr-2"
      onClick={this.removeTutorial}
    >
      Delete
    </button>

    <button
      type="submit"
      className="badge badge-success"
      onClick={this.updateContent}
    >
      Update
    </button>
    <p>{this.state.message}</p>
  </div>
) : (
  <div>
    <br />
    <p>Please click on a Tutorial...</p>
  </div>
)}
</div>
);
}
}

export default connect(null, { updateTutorial, deleteTutorial })(Tutorial);

```

Add CSS style for React Components

Open **src/App.css** and write some CSS code as following:

```

.list {
  text-align: left;
  max-width: 750px;
  margin: auto;
}

.submit-form {
  max-width: 300px;
  margin: auto;
}

.edit-form {
  max-width: 300px;
  margin: auto;
}

```

Configure Port for React Redux CRUD with Web API

Because most of HTTP Server use CORS configuration that accepts resource sharing restricted to some sites or ports, so we also need to configure port for our App.

In project folder, create `.env` file with following content:

```
PORT=8081
```

Now we've set our app running at port **8081**.

Run React Redux CRUD App

You can run our App with command: `npm start`.

If the process is successful, open Browser with Url: `http://localhost:8081/` and check it.

This React Client will work well with following back-end Rest APIs:

- Express, Sequelize & MySQL
- Express, Sequelize & PostgreSQL
- Express, Sequelize & SQL Server
- Express & MongoDB
- Spring Boot & MySQL
- Spring Boot & PostgreSQL
- Spring Boot & MongoDB
- Spring Boot & SQL Server
- Spring Boot & H2
- Spring Boot & Cassandra
- Spring Boot & Oracle
- Python/Django & MySQL
- Python/Django & PostgreSQL
- Python/Django & MongoDB

Conclusion

Today we've built a React Redux CRUD Application example successfully with React Router & Axios. Now we can consume REST APIs, display, search and modify data with Redux Store in a clean way. I hope you can make API call (GET/POST/PUT/DELETE) in your project at ease.

If you want to use Hooks instead:

React Hooks + Redux: CRUD example with Axios and Rest API

Implement Security:

- React JWT Authentication (without Redux) example
- React Redux: JWT Authentication example

Or you can add Pagination Component:

React Pagination with API using Material-UI

Tutorials List

Items per Page:

bezcoder Tut#19

bezcoder Tut#20

bezcoder Tut#21

Tutorial

Title: bezcoder Tut#20**Description:** Tut#20 Description**Status:** Published

Happy learning, see you again!

Further Reading

For more details about ways to use Axios, please visit:
Axios request: Get/Post/Put/Delete example

Fullstack:

- React Redux + Spring Boot example: CRUD example
- React + Spring Boot + MySQL: CRUD example
- React + Spring Boot + PostgreSQL: CRUD example
- React + Spring Boot + MongoDB: CRUD example
- React + Node.js + Express + MySQL: CRUD example
- React Redux + Node.js + Express + MySQL: CRUD example
- React + Node.js + Express + PostgreSQL example
- React + Node.js + Express + MongoDB example
- React + Django: CRUD example

If you want to implement Form Validation, please visit:
React Form Validation example

Source Code

You can find the complete source code for this example on Github.

Using Redux-Toolkit instead:
Redux-Toolkit example with CRUD Application