Working with Redux

Introduction:

- **State Management** is absolutely critical in a Web Application Development.
- We can manage state in a react application using **State** and **Context**.
- **State** contains data specific to a given component that may change over time.
- Using **Context**, we pass the data from parent component to Child component and from Child to Parent which are placed at different nesting levels.
- For low-frequency updates like **locale or theme** changes or **user authentication**, the **React Context** is perfectly fine.

Need of Redux:

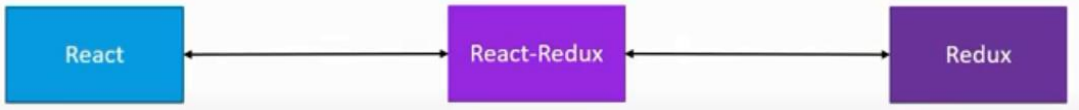
- But with a more complex state object like products in the shopping cart which has high-frequency updates, the React Context won't be a good solution.
- Because, the React Context will trigger a re-render on each update, and optimizing it manually can be really tough.
- **Redux** provides a solid, stable and mature solution to managing state in your React application.

What is Redux?

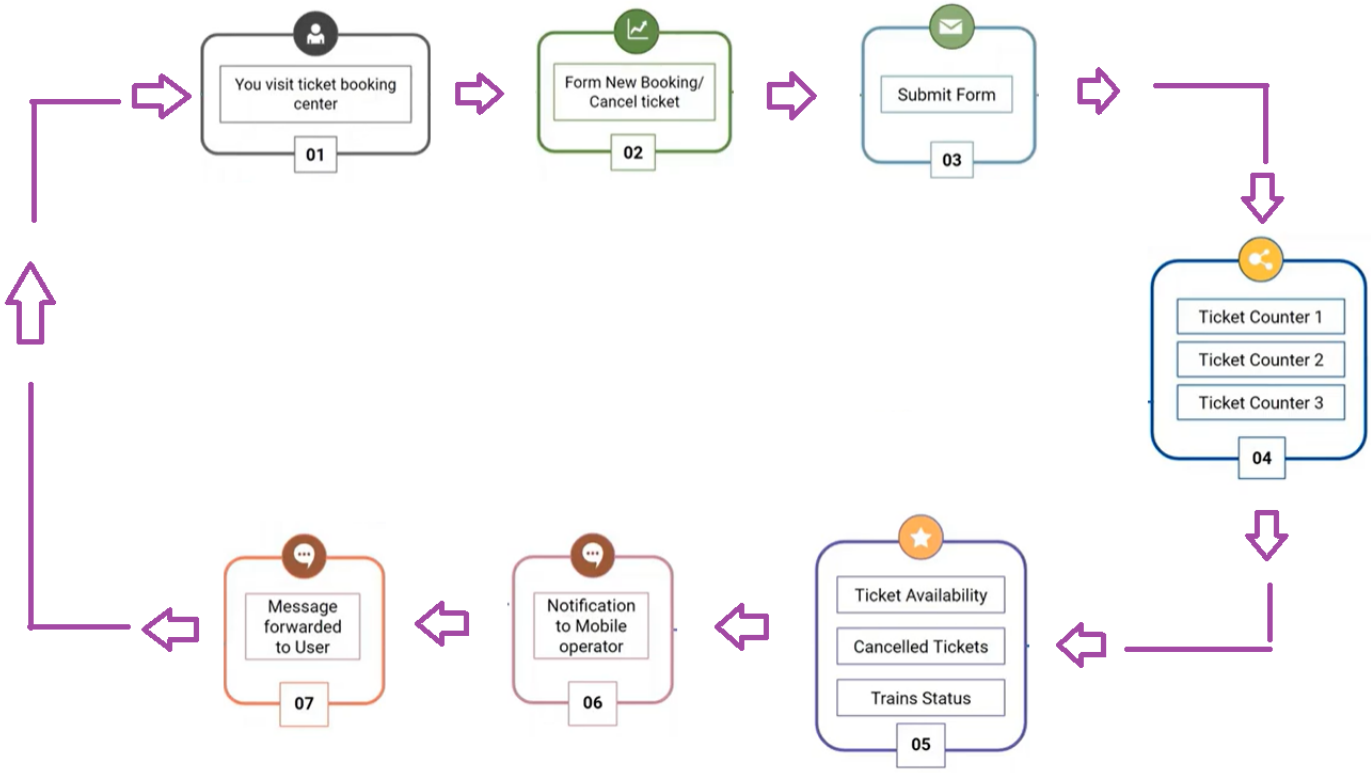
- Redux is a **State Management library** for JavaScript apps
- It provides **Centralized Store** where components can direct access the data.
- It is used to make **Complex Applications** easier and provide consistent data across application
- It's **flexible**, work with any UI frameworks like **React, Angular, Vue**, etc.
- **Easily Debuggable**, using **Redux Devtools** to trace when, where, why and how about application state.

Flux vs Redux vs React-Redux:

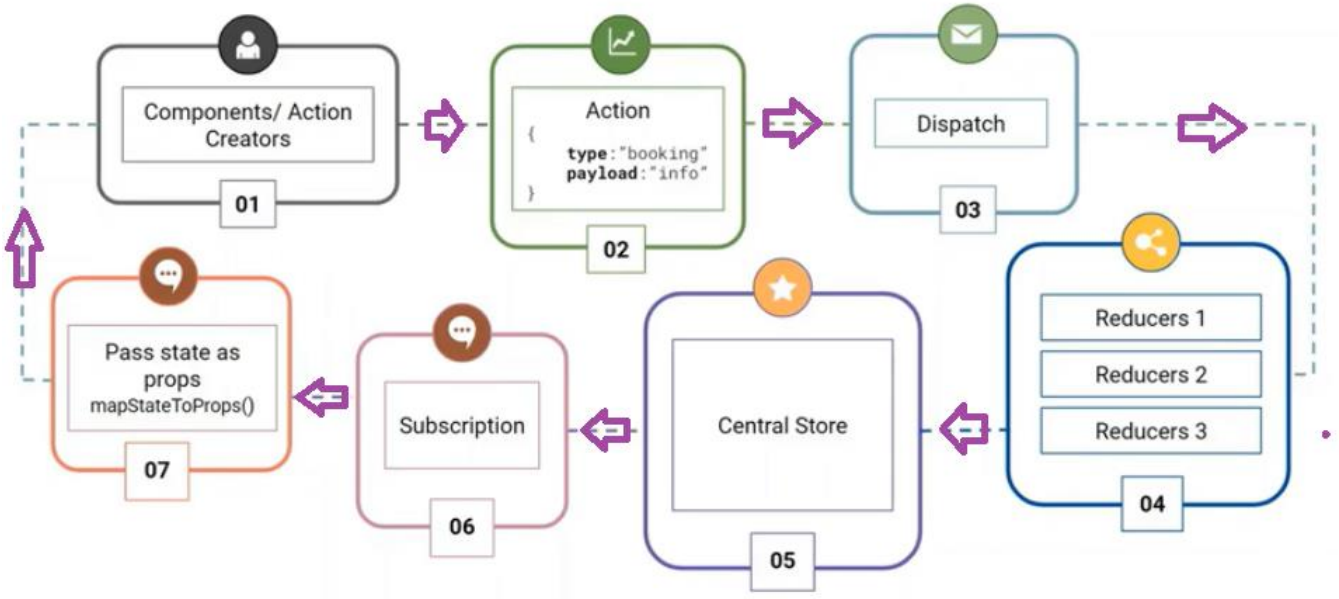
- **Flux** is an **architectural design pattern** that designed by Facebook in 2014.
- **Redux** is implementation of Flux
- **React-Redux** is a package which is implemented over Redux.



Railway Ticket Booking



Redux Analogy



Actions: (Event)

- An action is plain **JavaScript object** that has a **type and payload** properties.
- An action describes the changes in the state of the application.
- The **type** property should be string that indicates the type of action being performed.
- The **payload** property is an object which contains info.
- Types should typically be defined as string constants.

  Eg:

  Add Products to the store.

  Delete Products from Store.

Dispatch:

- Dispatch is a function of the Redux store.
- You call **store.dispatch** to dispatch an action.
- This is the only way to trigger a state change.

Reducer:

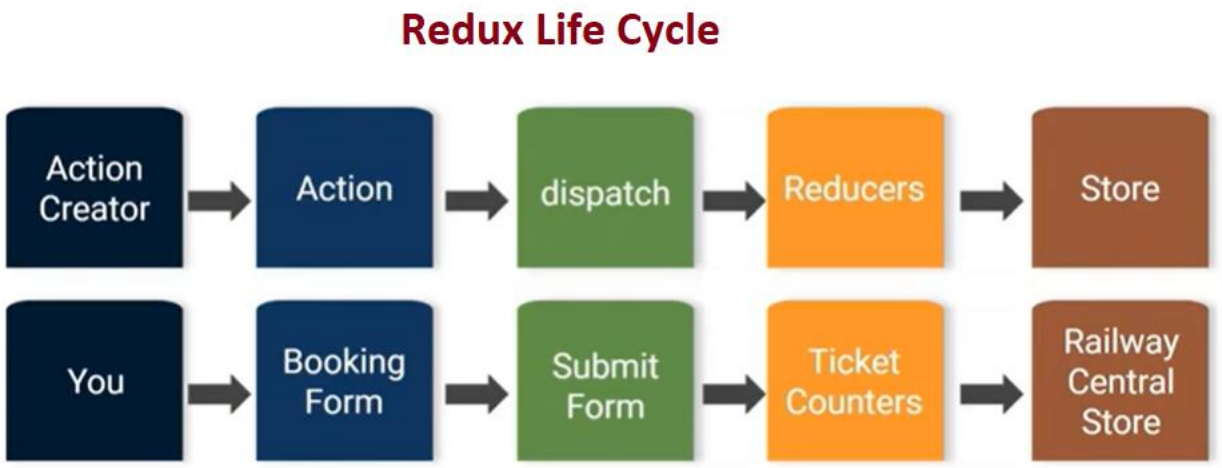
- A reducer is a function that receives the current **state** and an **action**, decides how to update the state if necessary, and returns the new state.

  (state, action) => newState.

Store:

- Store is **Centralized Store** which holds the state of application.
- The store is created using **createStore(reducer)** method by passing in a **reducer** as property.

  Eg: Online Store



Develop a ToDo App using react and redux

Create a new project.

C:\ReactJS>npx create-react-app todo-redux-app

Install redux and react-redux package.

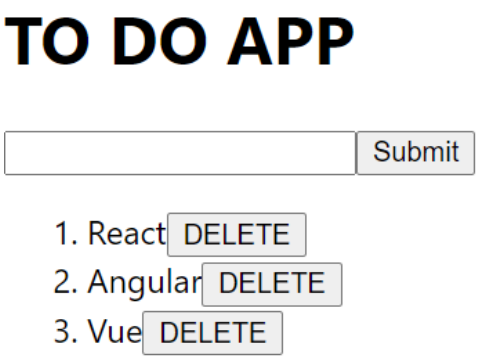
C:\ReactJS>cd todo-redux-app

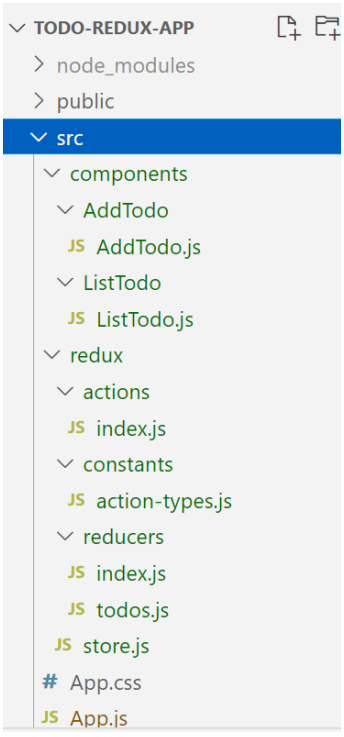
C:\ReactJS\todo-redux-app>npm install -save redux react-redux

Note:

React-Redux is a package which is implemented over Redux.

React-Redux is the official Redux UI binding library for React.





Create constants.

src\redux\constants\action-types.js

```
export const ActionTypes = {
  ADD_TODO: "ADD_TODO",
  DELETE_TODO: "DELETE_TODO",
};
```

Create actions.

src\redux\actions\index.js

```
import { ActionTypes } from "../constants/action-types";

export const addToDo = (message) => ({
  type: ActionTypes.ADD_TODO,
  message,
  id: Math.floor(Math.random() * 10),
});

export const deleteToDo = (id) => ({
  type: ActionTypes.DELETE_TODO,
  id,
});
```

Create a reducers

src\redux\reducers\todos.js

```
const initialState = {
  data: [],
};
const todos = (state = initialState, action) => {
  switch (action.type) {
    case "ADD_TODO":
      return {
        ...state,
        data: [
          ...state.data,
          {
            message: action.message,
            id: action.id,
          },
        ],
      };
    case "DELETE_TODO":
      const todos = state.data.filter((todo) => todo.id !== action.id);
      return {
```

```
        ...state,
        data: todos,
      };
    default:
      return state;
  }
};
export default todos;
```

Create rootReducers to combine all reducers.

src\redux\reducers\index.js

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

const rootReducer = combineReducers({
  todos,
});

export default rootReducer;
```

Create a store.

src\redux\store.js

```
import { createStore } from "redux";
import rootReducer from "./reducers";

const store = createStore(
  rootReducer,
);
export default store;
```

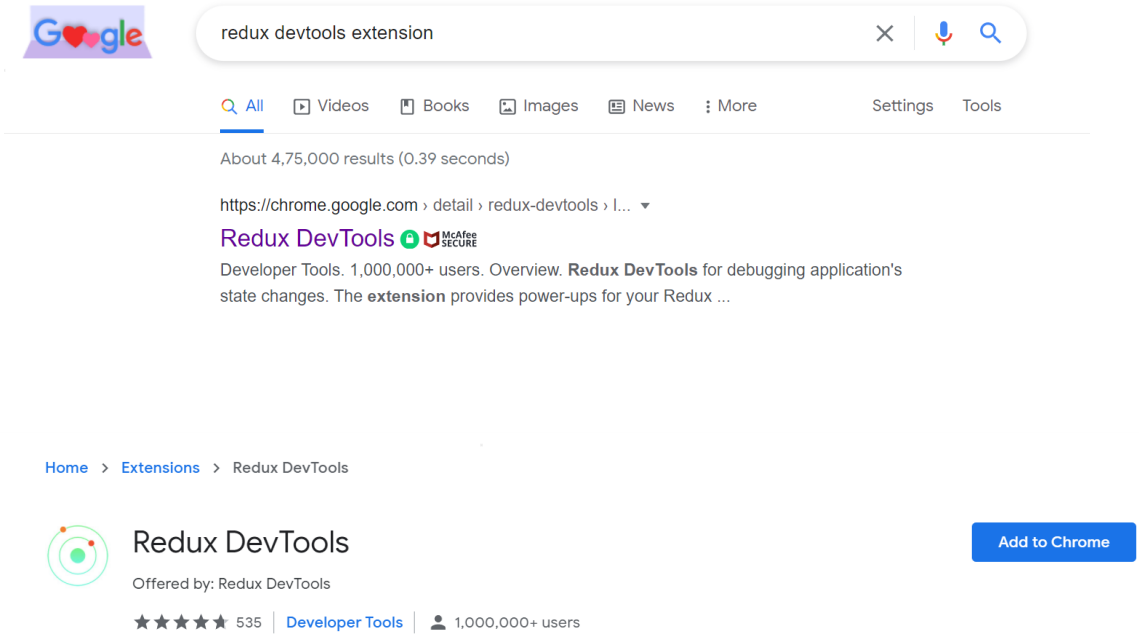
Configure the store in index.js

src\index.js

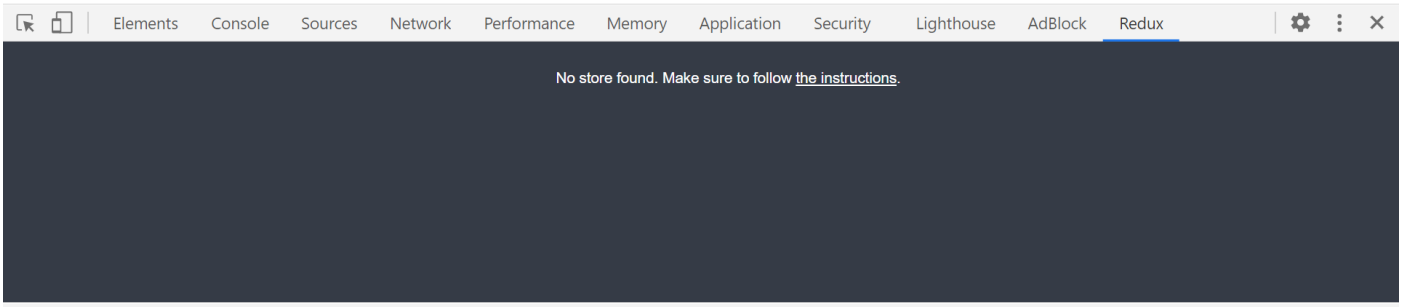
```
import React from "react";
import ReactDOM from "react-dom";
import "./index.css";
import App from "./App";
import store from "./redux/store";
import { Provider } from "react-redux";

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

Integrate Redux Developer Tools in Google Chrome.



Inspect the home page on Google chrome and open *Redux* tab.



Click on *the instructions*

1. With Redux

1.1 Basic store

For a basic Redux store simply add:

```
 const store = createStore(
   reducer, /* preloadedState, */
+  window.__REDUX_DEVTOOLS_EXTENSION__ && window.__REDUX_DEVTOOLS_EXTENSION__()
 );
```

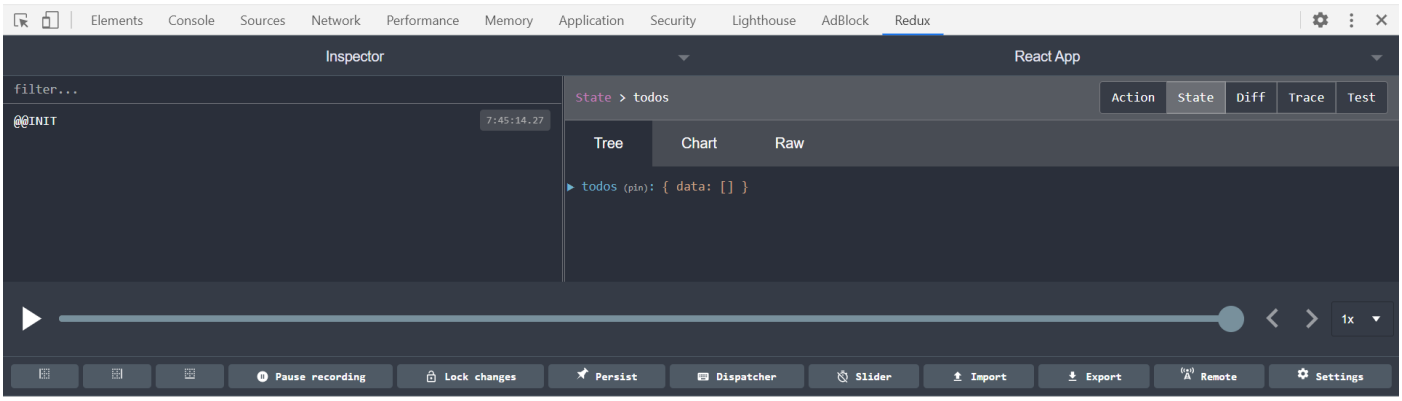
Update store.js file

src\redux\store.js

```
import { createStore } from "redux";
import rootReducer from "./reducers";

const store = createStore(
  rootReducer,
  window.__REDUX_DEVTOOLS_EXTENSION__ && window.__REDUX_DEVTOOLS_EXTENSION__()
);

export default store;
```



Create AddToDo component.

src\components\AddTodo\AddTodo.js

```
import React from "react";
import { connect } from "react-redux";
import { addToDo } from "../../redux/actions";

const AddTodo = (props) => {
  const onSubmitHandler = (event) => {
    event.preventDefault();
    let input = event.target.userInput.value;
    console.log(input);
    props.dispatch(addToDo(input));
    event.target.userInput.value = "";
  };

  return (
    <div>
      <form onSubmit={onSubmitHandler}>
        <input type="text" name="userInput"></input>
        <button>Submit</button>
      </form>
    </div>
  );
};

export default connect()(AddTodo);
```

Create ListToDo component.

src\components\ListTodo\ListTodo.js

```
import React from "react";
import { connect } from "react-redux";
import { deleteToDo } from "../../redux/actions";

const ListTodo = (props) => {
  return (
    <div>
      <ol>
        {props.todos.map((todo, index) => (
          <li key={index}>
            {todo.message}
            <button onClick={() => props.dispatch(deleteToDo(todo.id))}>
              DELETE
            </button>
          </li>
        ))}
      </ol>
    </div>
```

```
  );
};

const mapStateToProps = (state) => ({
  todos: state.todos.data,
});

export default connect(mapStateToProps)(ListTodo);
```

src\App.js

```
import "./App.css";
import AppTodo from "./components/AddTodo/AddTodo";
import ListTodo from "./components/ListTodo/ListTodo";

function App() {
  return (
    <div>
      <h2>TO DO APP</h2>
      <AppTodo></AppTodo>
      <ListTodo></ListTodo>
    </div>
  );
}

export default App;
```

TO DO APP

Submit

1. Task1 DELETE
2. Task2 DELETE