SESSION 5

REDUX

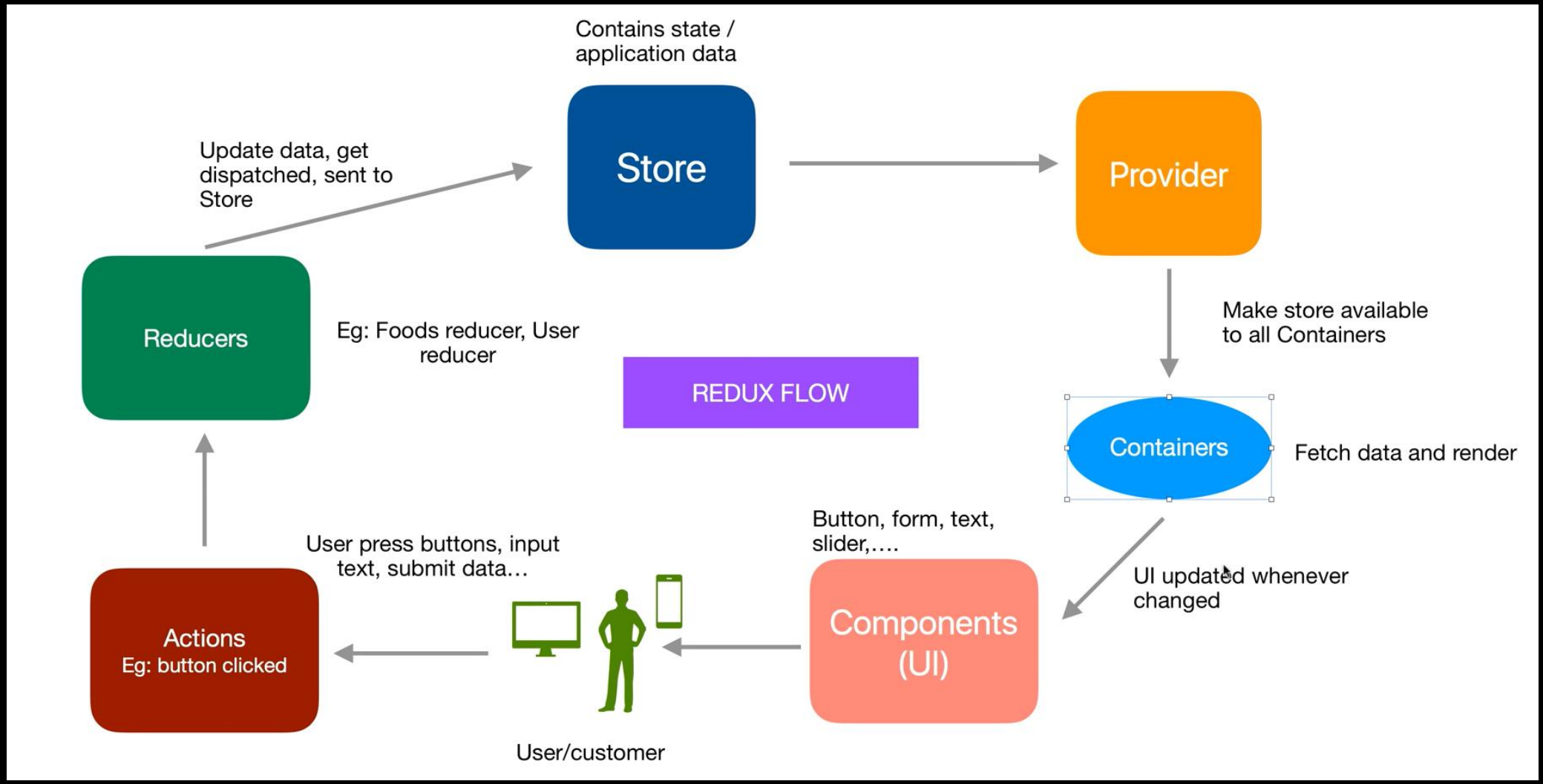
Aptech™
COMPUTER EDUCATION
Unleash your potential
Alliance with FPT Education

OBJECTIVES

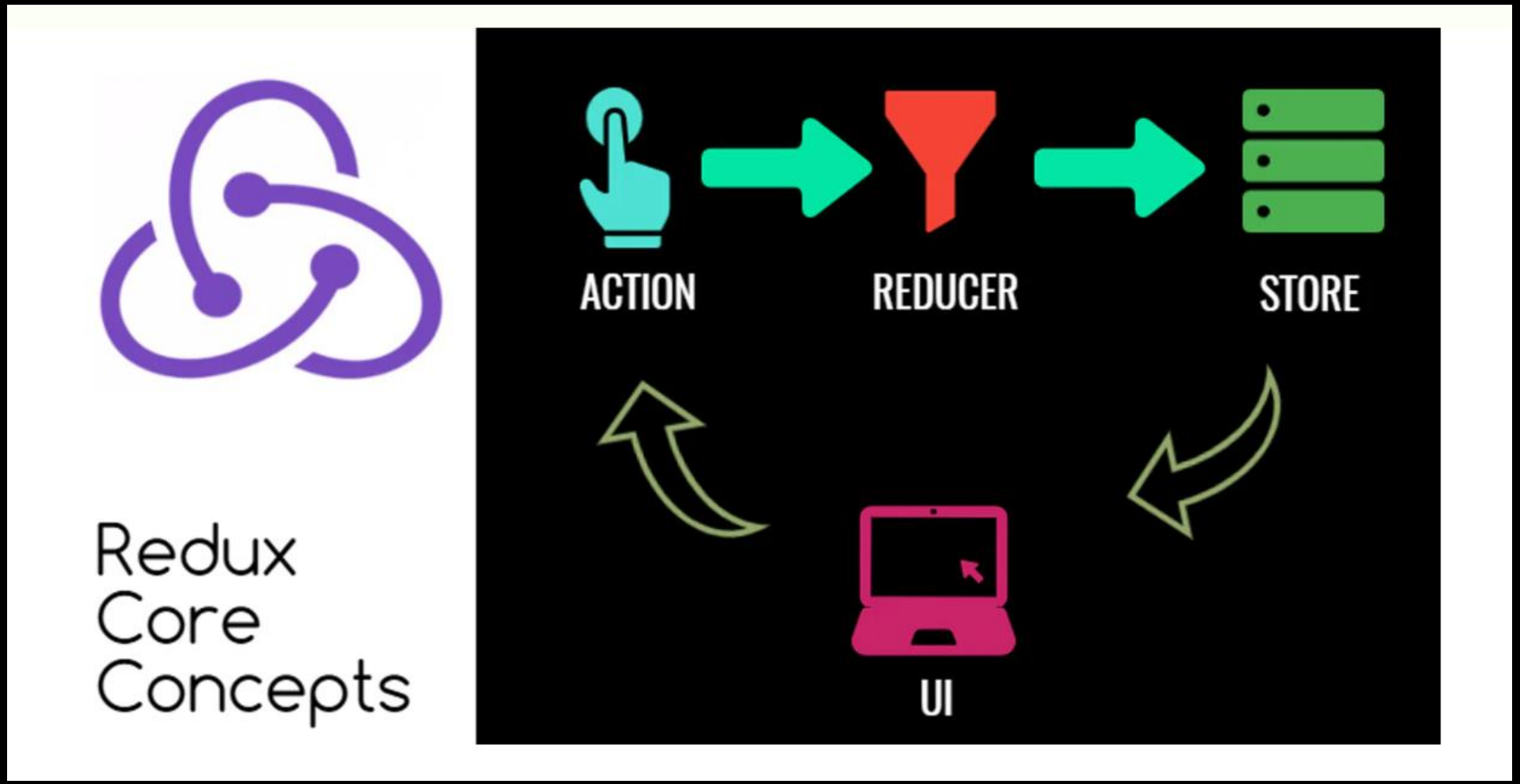
What's Redux?
Why I should use Redux?
Terminology of Redux :

- Actions
- Reducers
- Store
- Dispatch
- Connect

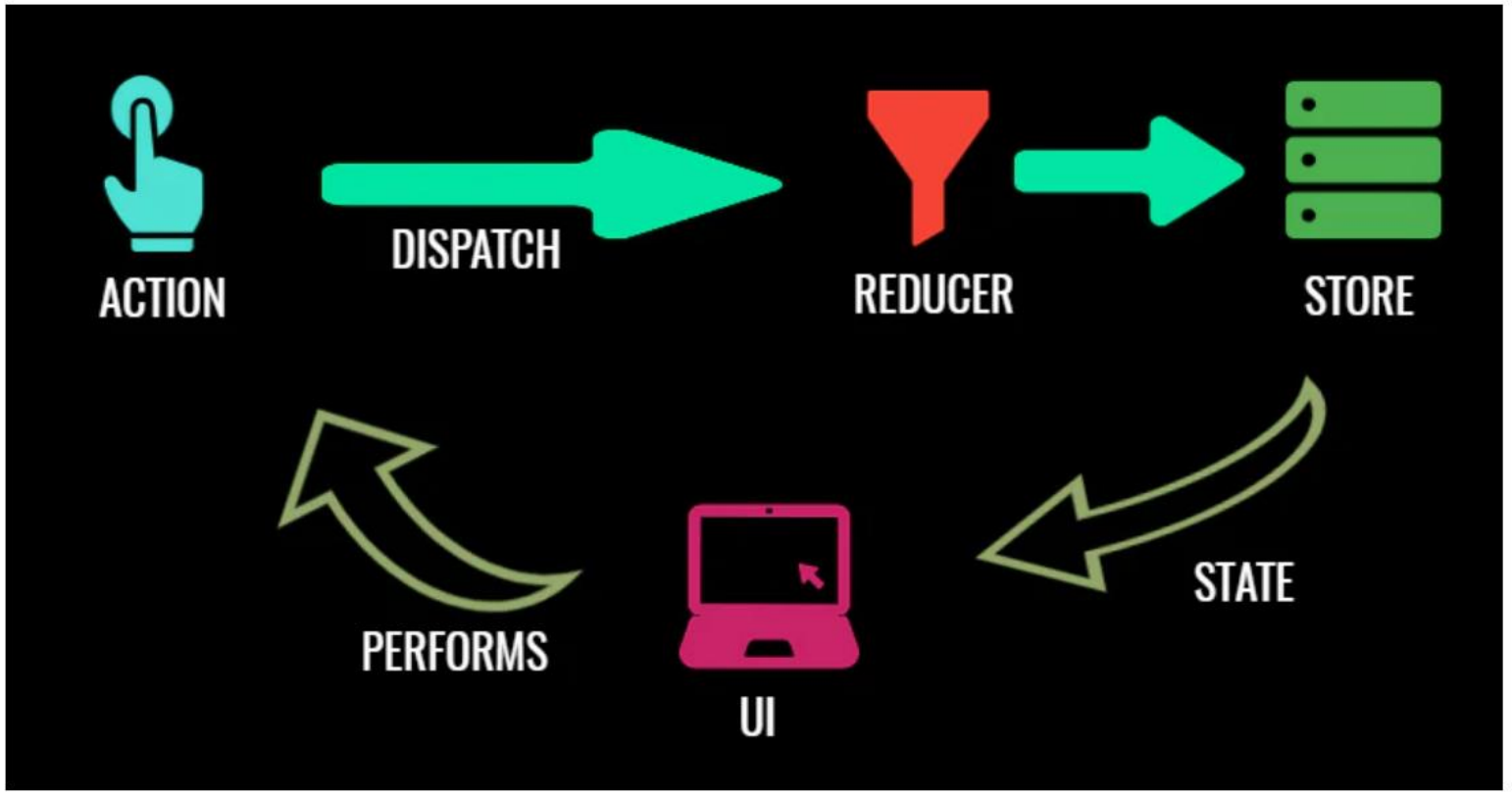
REDUX FLOW


Contains state / application data
Store
Provider
Update data, get dispatched, sent to Store
Reducers
Eg: Foods reducer, User reducer
REDUX FLOW
Make store available to all Containers
Containers
Fetch data and render
Button, form, text, slider,….
User press buttons, input text, submit data…
UI updated whenever changed
Actions
Eg: button clicked
Components (UI)
User/customer

CONCEPTS



STATE MANAGEMENT



State management using Redux

WHAT IS REDUX?

- Redux is a **state container** for JavaScript applications.
- **Normally with React**, **you manage state at a component level, and pass state around via props.**
- With Redux, **the entire state of your application is managed** in one immutable object. Every update to the Redux state results in a copy of sections of the state, plus the new change.
- **Redux is a pattern and library for managing and updating application state, using events called "actions**".

WHY SHOULD I USE REDUX

- **Easily manage global state**
- **Easily keep track of changes with Redux DevTools**

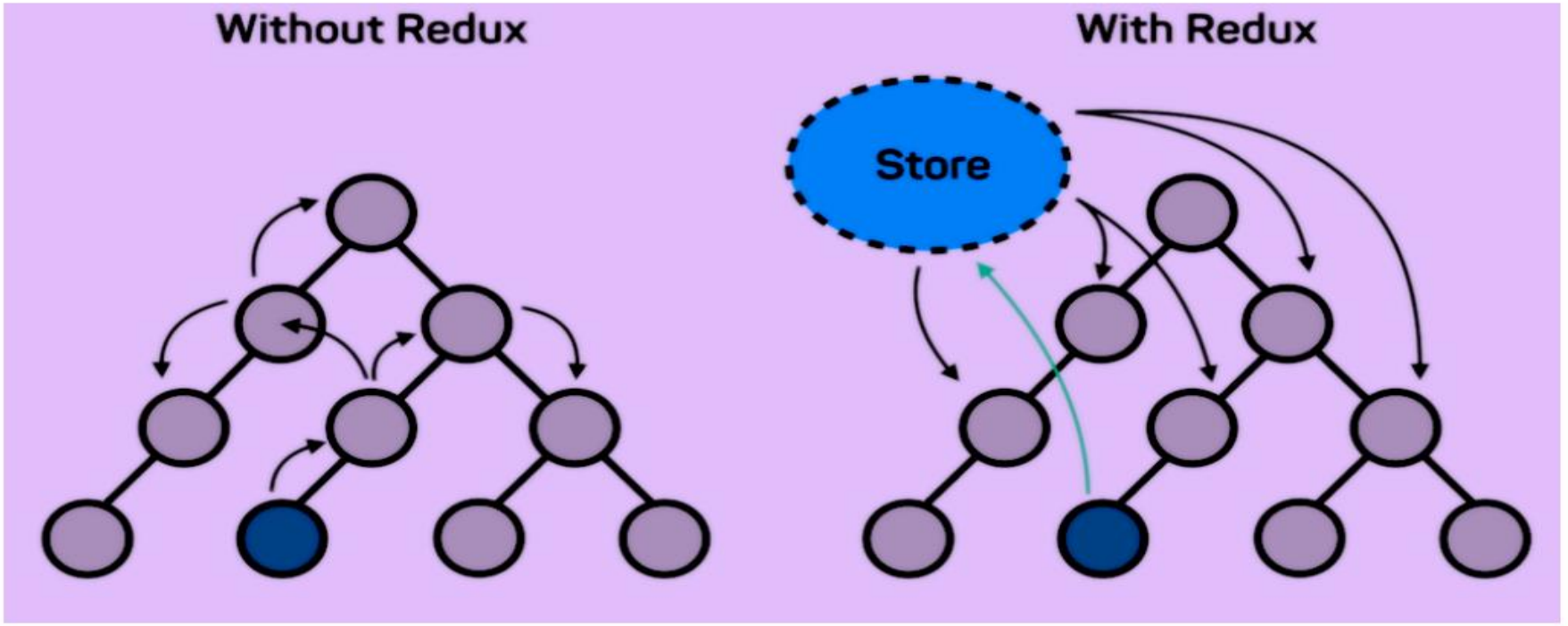
ADVANTAGES OF REDUX

- Predictability of outcome
- Maintainability
- Server side rendering
- Developer tools
- Community and ecosystem
- Ease of testing
- Organization

WHY REDUX?

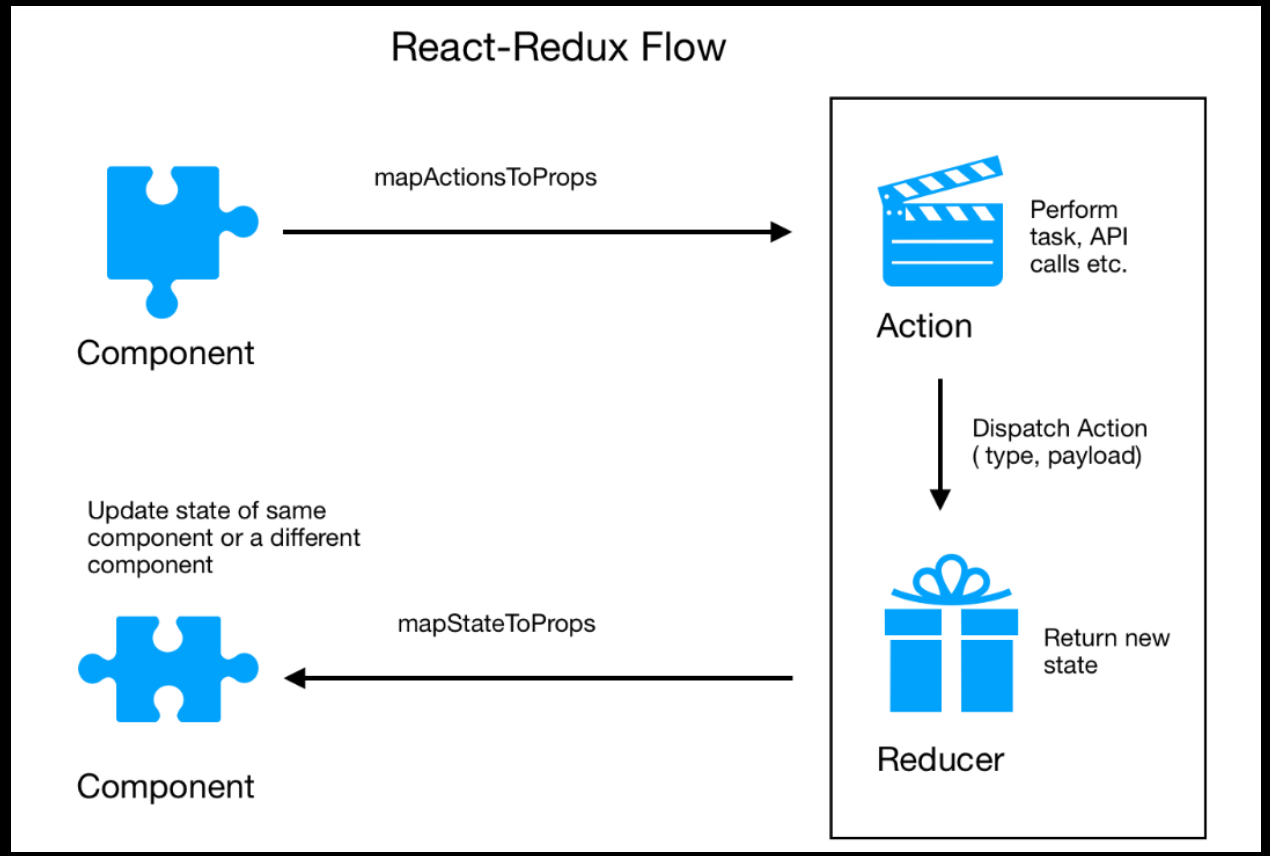
Why Redux?

Observe the infamous diagram below:



Two Application Examples with Components passing State

REDUX FLOW


React-Redux Flow
mapActionsToProps
Component
Perform task, API calls etc.
Action
Dispatch Action (type, payload)
Update state of same component or a different component
mapStateToProps
Return new state
Reducer
Component

SIMPLE REDUX CREATE DELETE CONTACT APPLICATION

Aptech
COMPUTER EDUCATION
Unleash your potential
Alliance with FPT Education

Redux is the most famous library to manage the state at the client side, and it is trendy among React community.

We will build a Simple Contact Create Delete Application.

In that user can add a new contact and it will display as a list and user can also delete any contact.

REDUX IS MORE USEFUL

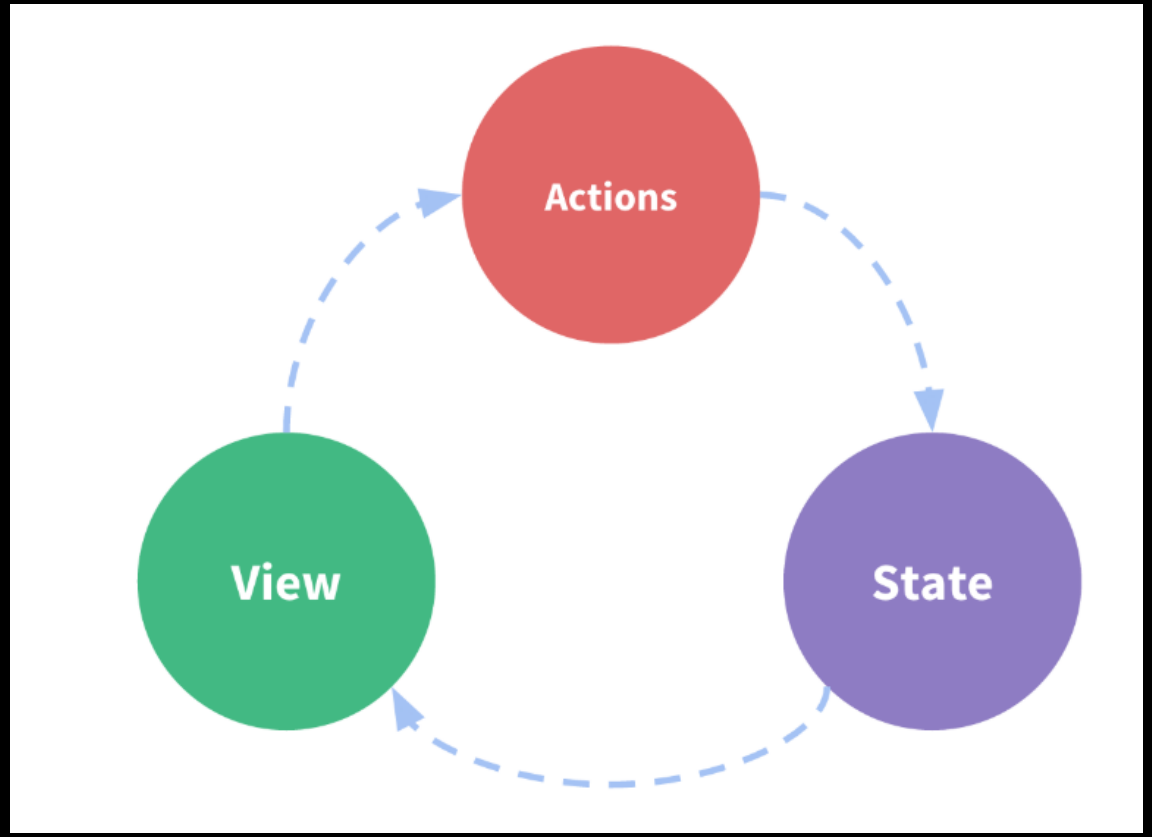
- You have large amounts of application state that are needed in many places in the app
- The app state is updated frequently over time
- The logic to update that state may be complex
- The app has a medium or large-sized codebase, and might be worked on by many people

TERMINOLOGY

- Actions
- Reducers
- Store
- Dispatch
- Connect

TERMINOLOGY

The state, the source of truth that drives our app;
The view, a declarative description of the UI based on the current state
The actions, the events that occur in the app based on user input, and trigger updates in the state



ACTIONS

An action sends data from your application to the Redux store. An action is conventionally an object with two properties: type and (optional) payload.

The type is generally an uppercase string (assigned to a constant) that describes the action. The payload is additional data that may be passed.

ACTIONS

- **Action Type** :

```
const DELETE_TODO = 'DELETE_TODO'
```

Action :

```
{
  type: DELETE_TODO,
  payload: id,
}
```

Action creators:

An **action creator** is a function that returns an action

REDUCER

- **A reducer is a function** that takes two parameters: **state and action**.
- **A reducer is immutable** and always returns a copy of the entire state.
- **A reducer** typically consists of **a switch statement** that goes through all the possible action types.

STORE

- **The Redux application state lives in the store**, which **is initialized with a reducer**.
- When used with React, **a <Provider> exists** to wrap the application, and anything within the Provider can have access to Redux.

DISPATCH

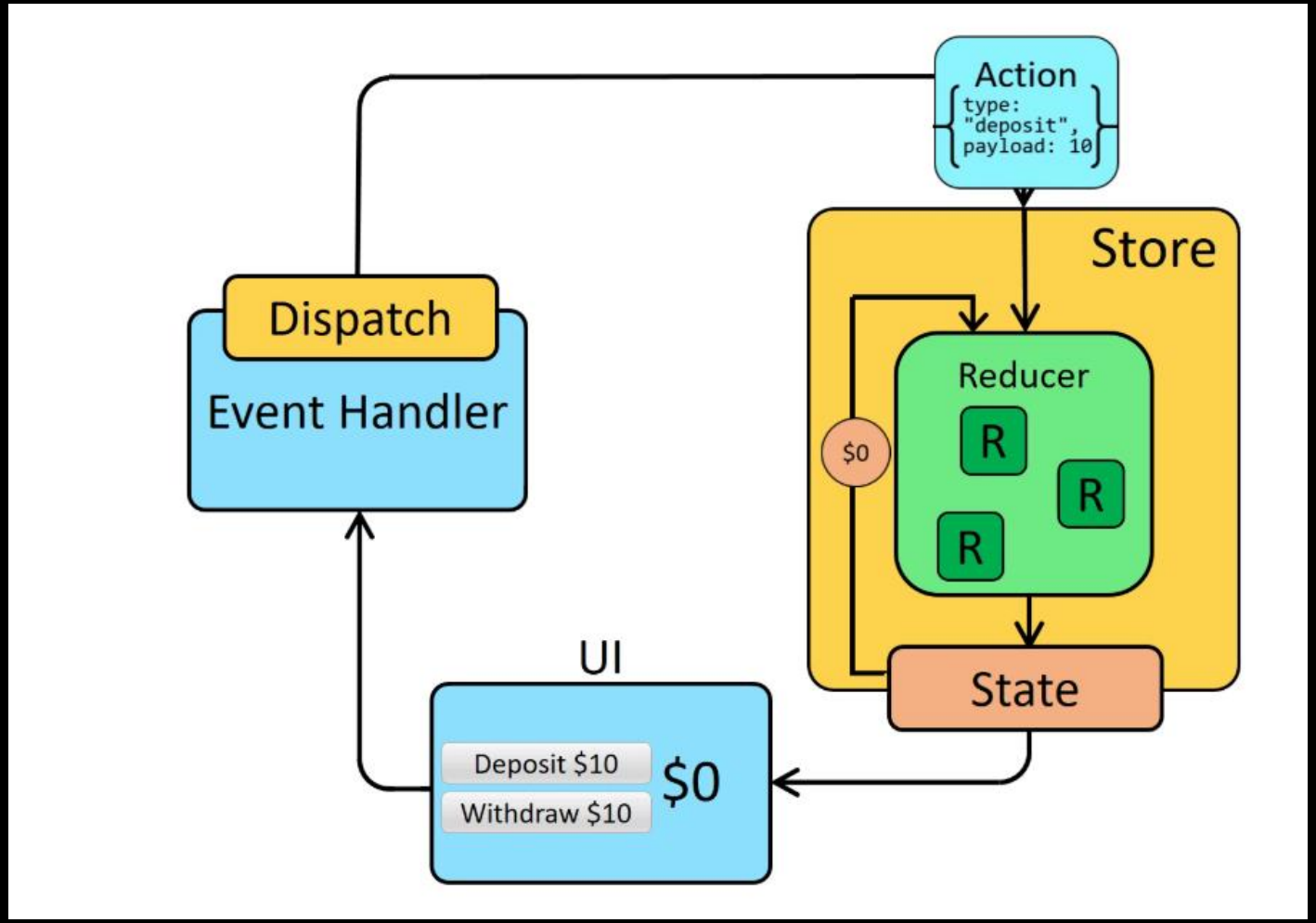
- Dispatch is a method available on the store object that accepts an object which is used to update the Redux state. Usually, this object is the result of invoking an action creator.

```
const Component = ({dispatch}) => {
        useEffect(() =>  {
                dispatch(deleteTodo())
          }, [dispatch])
        }
```

CONNECT

- The connect() function is one typical way to connect React to Redux. A connected component is sometimes referred to as a container.

DATA FLOW

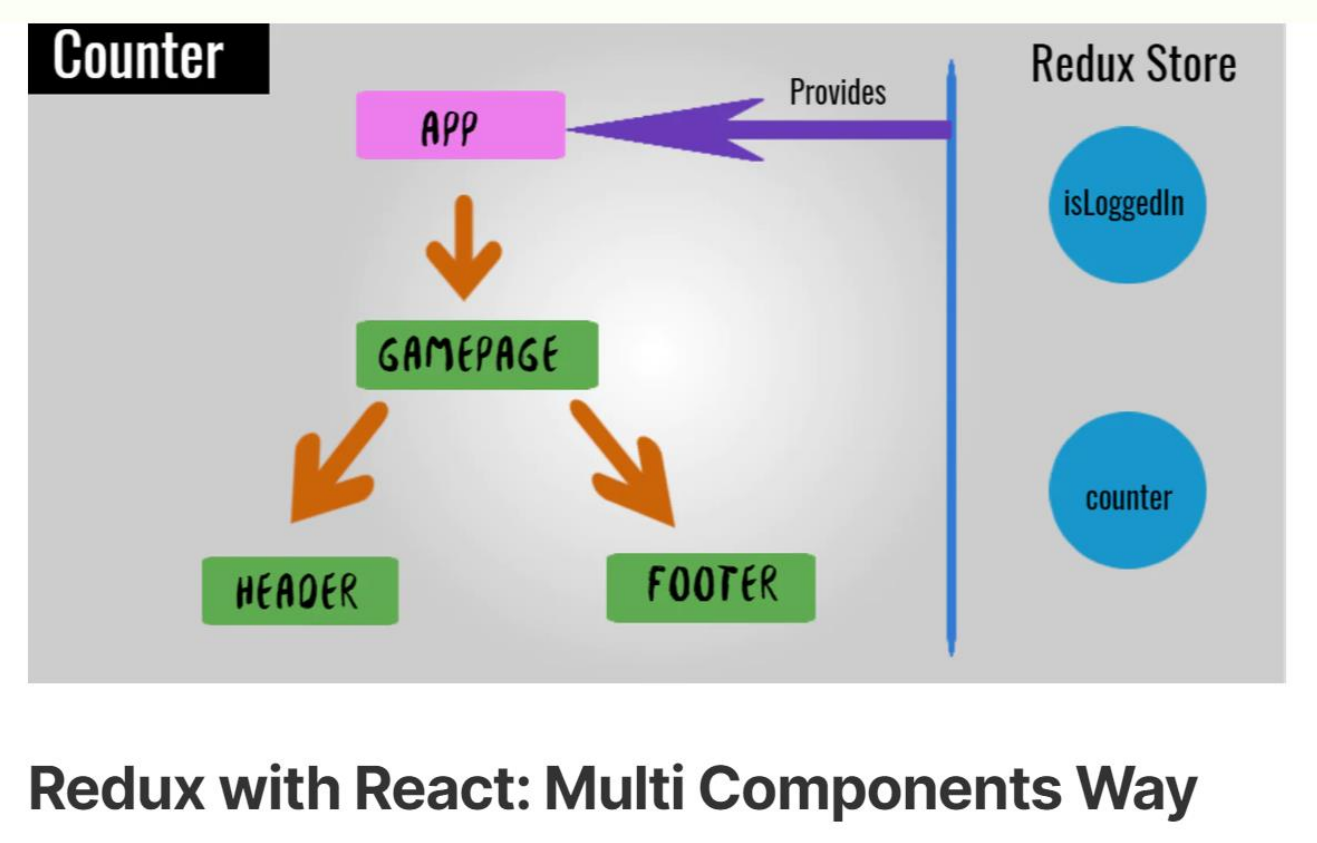

Action
type:
"deposit",
payload: 10
Store
Dispatch
Event Handler
Reducer
R
R
R
$0
State
UI
Deposit $10
Withdraw $10
$0

Aptech COMPUTER EDUCATION Unleash your potential
Alliance with FPT Education

```
const initialState = {
  todos: [
    {id: 1, text: 'Eat'},
    {id: 2, text: 'Sleep'},
  ],
  loading: false,
  hasErrors: false,
}

function todoReducer(state = initialState, action) {
  switch (action.type) {
    case DELETE_TODO:
      return {
        ...state,
        todos: state.todos.filter((todo) => todo.id !== action.payload),
      }
    default:
      return state
  }
}
```

MULTI COMPONENTS



DEMO REDUX COUNTER APP

- Step 1 : Create React app

  npx create-react-app count-redux-react
- Step 2 : npm add redux react-redux –save
- Step 3 : package.json

```
"dependencies": {
  "react": "^16.10.2",
  "react-dom": "^16.10.2",
  "react-redux": "^7.1.1",
  "react-scripts": "3.2.0",
  "redux": "^4.0.4"
},
```

DEMO REDUX COUNTER APP

- Step 4 : App.js

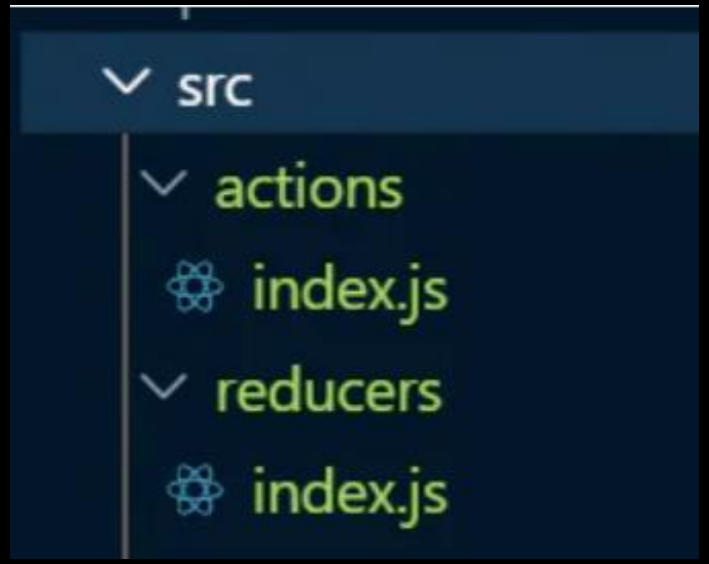
```
import React from 'react';

function App() {
  return (
    <div className="App">
      <h1> Blend Redux with ReactJS </h1>
    </div>
  );
}

export default App;
```

DEMO REDUX COUNTER APP

- Step 5 : Create two folders actions and reducers



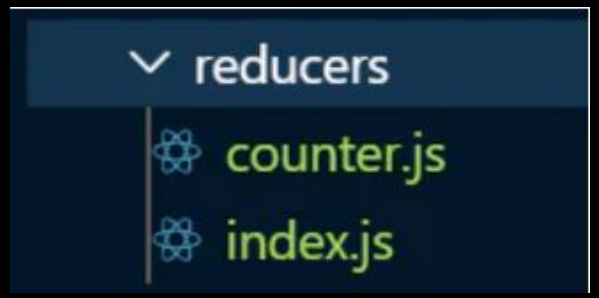
Step 6: Modify index.js (under the actions folder)

```
export const increment = () => {
    return {
        type: 'INCREMENT'
    }
};

export const decrement = () => {
    return {
        type: 'DECREMENT'
    }
};
```

DEMO REDUX COUNTER APP

Step 7: Create file counter.js




```
const counterReducer = (state = 0, action) => {
    switch(action.type) {
        case 'INCREMENT':
            return state + 1;
        case 'DECREMENT':
            return state -1;
        default:
            return state;
    }
}

export default counterReducer;
```

DEMO REDUX COUNTER APP

- Step 8: Modify the index.js (under reducers)

```
import counterReducer from './counter';
import { combineReducers } from 'redux';

const allReducers = combineReducers({
    counter: counterReducer
});
```

```
const allReducers = combineReducers({
    counter: counterReducer,
    user: userReducer
});
```

DEMO REDUX COUNTER APP

Aptech COMPUTER EDUCATION Unleash your potential Alliance with FPT Education

- Step 9: Create Redux Store:

```
import { createStore } from 'redux';
import allReducers from './reducers';

const store = createStore(allReducers);
```

DEMO REDUX COUNTER APP

- Step 10: modify index.js (App)

```
import { Provider } from 'react-redux';

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

DEMO REDUX COUNTER APP

- Step 11: in App.js , add 2 buttons

```
return (
    <div className="App">
      <h1> Blend Redux with ReactJS</h1>
      <h1>Counter: 0</h1>
      <button>+</button>
      <button>-</button>
    </div>
  );
```

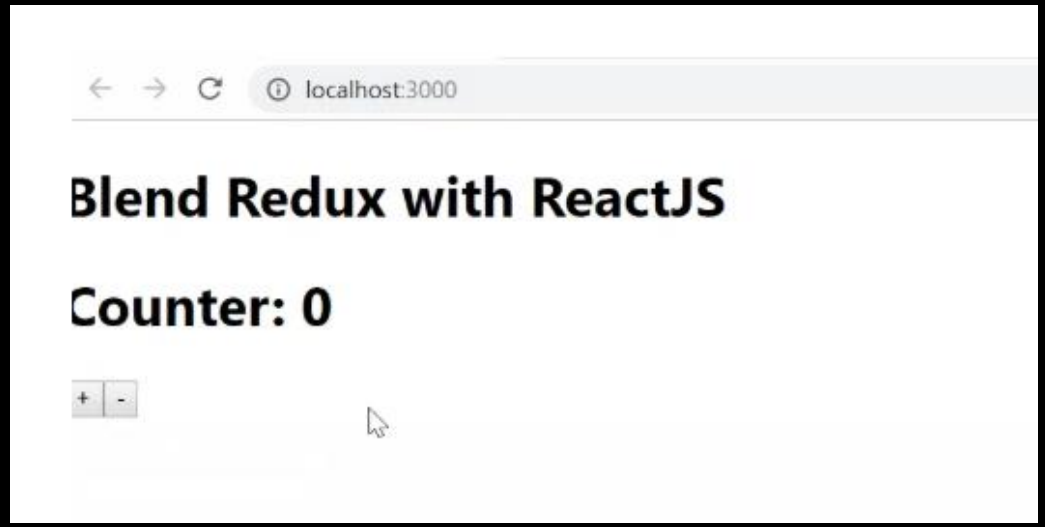
DEMO REDUX COUNTER APP

Step 12 : App.js

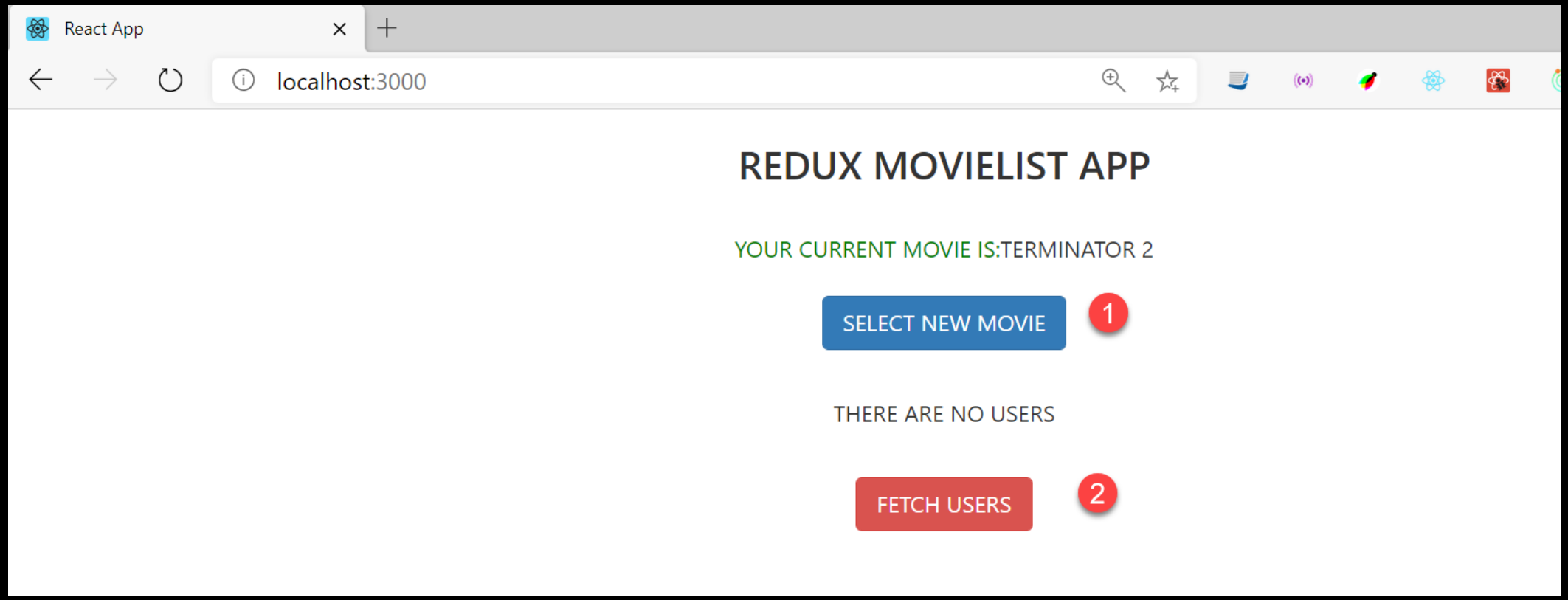
```
import { increment, decrement} from './actions';
import { useSelector, useDispatch } from 'react-redux';

function App() {
  const counter = useSelector(state => state.counter);
  const dispatch = useDispatch();
  return (
    <div className="App">
      <h1> Blend Redux with ReactJS</h1>
      <h1>Counter: {counter}</h1>
      <button onClick={() => dispatch(increment())}>+</button>
      <button onClick={() => dispatch(decrement())}>-</button>
    </div>
  );
}
```

Step 13: Run



LAB : REACT AND REDUX FRONT-END MOVIE APP DEMO WITH API FETCH



THE END

Aptech™
COMPUTER EDUCATION
Unleash your potential
Alliance with FPT Education