13. REACT Flux

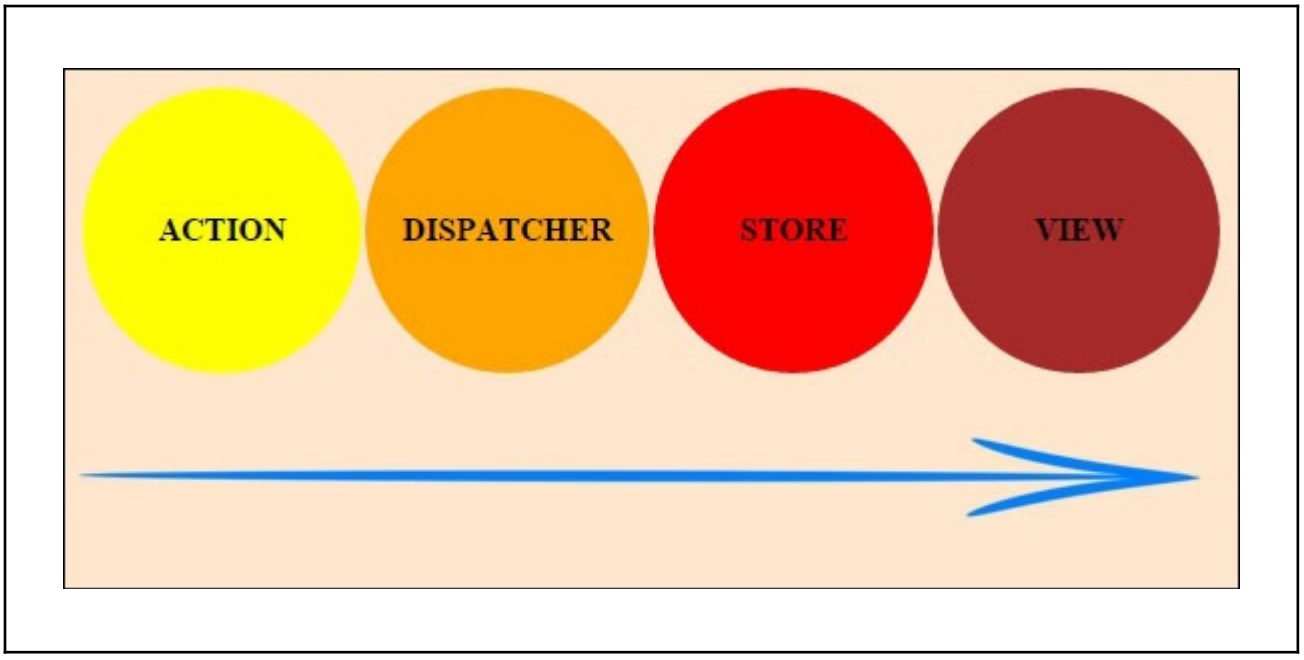
Flux is a programming concept, where the data is uni-directional. This data enters the app and flows through it in one direction until it is rendered on the screen.

13.1. Flux Elements

Following is a simple explanation of the flux concept. In the next chapter, we will learn how to implement this into the app.

- **Actions** − Actions are sent to the dispatcher to trigger the data flow.
- **Dispatcher** − This is a central hub of the app. All the data is dispatched and sent to the stores.
- **Store** − Store is the place where the application state and logic are held. Every store is maintaining a particular state and it will update when needed.
- **View** − The view will receive data from the store and re-render the app.

The data flow is depicted in the following image.



Flux Pros

- Single directional data flow is easy to understand.
- The app is easier to maintain.
- The app parts are decoupled.

13.2. Using Flux

In this chapter, we will learn how to implement flux pattern in React applications. We will use **Redux** framework. The goal of this chapter is to present the simplest example of every piece needed for connecting **Redux** and React.

Step 1 - Install Redux

We will install Redux via the command prompt window.

Example:

```
C:\Users\MyUser\Desktop\reactApp>npm install --save react-redux
```

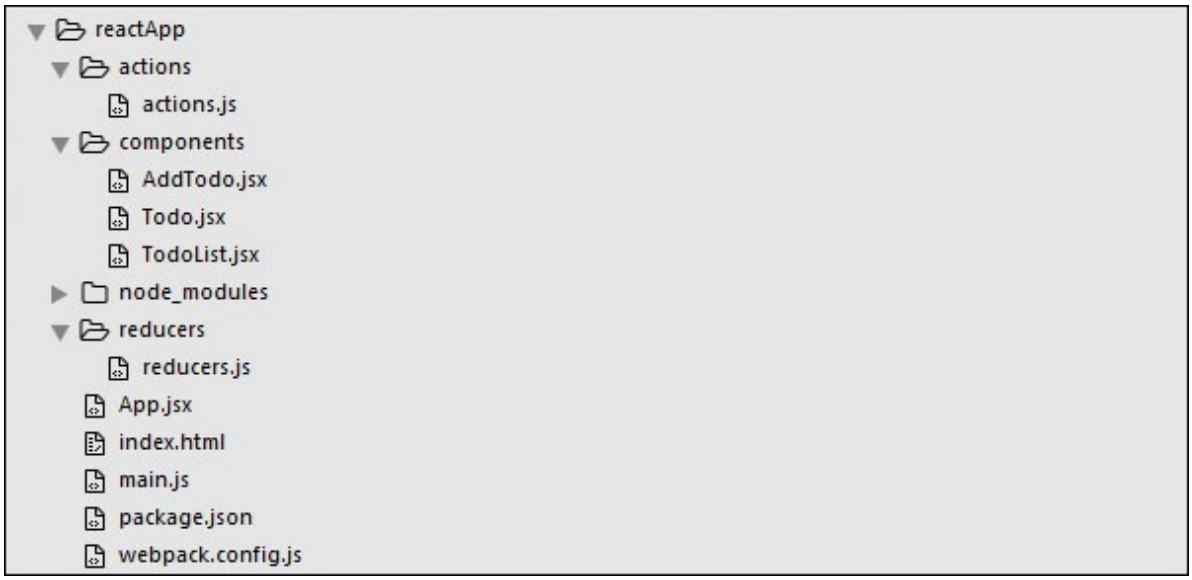
Step 2 - Create Files and Folders

In this step, we will create folders and files for our actions, reducers, and components. After we are done with it, this is how the folder structure will look like.

Example:

```
C:\Users\MyUser\Desktop\reactApp>mkdir actions
C:\Users\MyUser\Desktop\reactApp>mkdir components
C:\Users\MyUser\Desktop\reactApp>mkdir reducers
C:\Users\MyUser\Desktop\reactApp>type nul > actions/actions.js
C:\Users\MyUser\Desktop\reactApp>type nul > reducers/reducers.js
C:\Users\MyUser\Desktop\reactApp>type nul > components/AddTodo.js
C:\Users\MyUser\Desktop\reactApp>type nul > components/Todo.js
C:\Users\MyUser\Desktop\reactApp>type nul > components/TodoList.js
```

Output:



Step 3 - Actions

Actions are JavaScript objects that use **type** property to inform about the data that should be sent to the store. We are defining **ADD_TODO** action that will be used for adding new item to our list. The **addTodo** function is an action creator that returns our action and sets an **id** for every created item.

Example: actions/actions.js

```
export const ADD_TODO = 'ADD_TODO'

let nextTodoId = 0;

export function addTodo(text) {
   return {
      type: ADD_TODO,
      id: nextTodoId++,
```

```
            text
        };
    }
```

Step 4 – Reducers

While actions only trigger changes in the app, the reducers specify those changes. We are using switch statement to search for a ADD_TODO action. The reducer is a function that takes two parameters (state and action) to calculate and return an updated state.

The first function will be used to create a new item, while the second one will push that item to the list. Towards the end, we are using combineReducers helper function where we can add any new reducers we might use in the future.

Example: reducers/reducers.js

```
import { combineReducers } from 'redux'
import { ADD_TODO } from '../actions/actions'

function todo(state, action) {
   switch (action.type) {
      case ADD_TODO:
         return {
            id: action.id,
            text: action.text,
         }
      default:
         return state
   }
}
function todos(state = [], action) {
   switch (action.type) {
```

```
        case ADD_TODO:
            return [
                ...state,
                todo(undefined, action)
            ]
        default:
            return state
    }
}
const todoApp = combineReducers({
    todos
})
export default todoApp
```

Step 5 – Store

The store is a place that holds the app's state. It is very easy to create a store once you have reducers. We are passing store property to the **provider** element, which wraps our route component.

Example: main.js

```
import React from 'react'

import { render } from 'react-dom'
import { createStore } from 'redux'
import { Provider } from 'react-redux'

import App from './App.jsx'
import todoApp from './reducers/reducers'

let store = createStore(todoApp)
```

```
let rootElement = document.getElementById('app')

render(
   <Provider store = {store}>
      <App />
   </Provider>,

   rootElement
)
```

Step 6 – Root Component

The *App* component is the root component of the app. Only the root component should be aware of a redux. The important part to notice is the connect function which is used for connecting our root component App to the store.

This function takes **select** function as an argument. Select function takes the state from the store and returns the props (*visibleTodos*) that we can use in our components.

Example: App.jsx

```
import React, { Component } from 'react'
import { connect } from 'react-redux'
import { addTodo } from './actions/actions'

import AddTodo from './components/AddTodo.js'
import TodoList from './components/TodoList.js'

class App extends Component {
   render() {
      const { dispatch, visibleTodos } = this.props
```

```
        return (
            <div>
                <AddTodo onAddClick = {text
=>dispatch(addTodo(text))} />
                <TodoList todos = {visibleTodos}/>
            </div>
        )
    }
}
function select(state) {
    return {
        visibleTodos: state.todos
    }
}
export default connect(select)(App);
```

Step 7 – Other Components

These components shouldn't be aware of redux.

Example: components/AddTodo.js

```
import React, { Component, PropTypes } from 'react'

export default class AddTodo extends Component {
    render() {
        return (
            <div>
                <input type = 'text' ref = 'input' />

                <button onClick = {(e) => this.handleClick(e)}>
```

```
                    Add
                </button>
            </div>
        )
    }
    handleClick(e) {
        const node = this.refs.input
        const text = node.value.trim()
        this.props.onAddClick(text)
        node.value = ''
    }
}
```

Example: components/Todo.js

```
import React, { Component, PropTypes } from 'react'

export default class Todo extends Component {
    render() {
        return (
            <li>
                {this.props.text}
            </li>
        )
    }
}
```

Example: components/TodoList.js

```
import React, { Component, PropTypes } from 'react'
import Todo from './Todo.js'

export default class TodoList extends Component {
   render() {
      return (
         <ul>
            {this.props.todos.map(todo =>
               <Todo
                  key = {todo.id}
                  {...todo}
               />
            )}
         </ul>
      )
   }
}
```

When we start the app, we will be able to add items to our list.

Output: