

Umberto Emanuele

Redux – Lezione 2


Gennaio 2023



🔗 Installing React Redux

[React bindings](#) are not included in Redux by default. You need to install them explicitly:

```
npm install --save react-redux
```

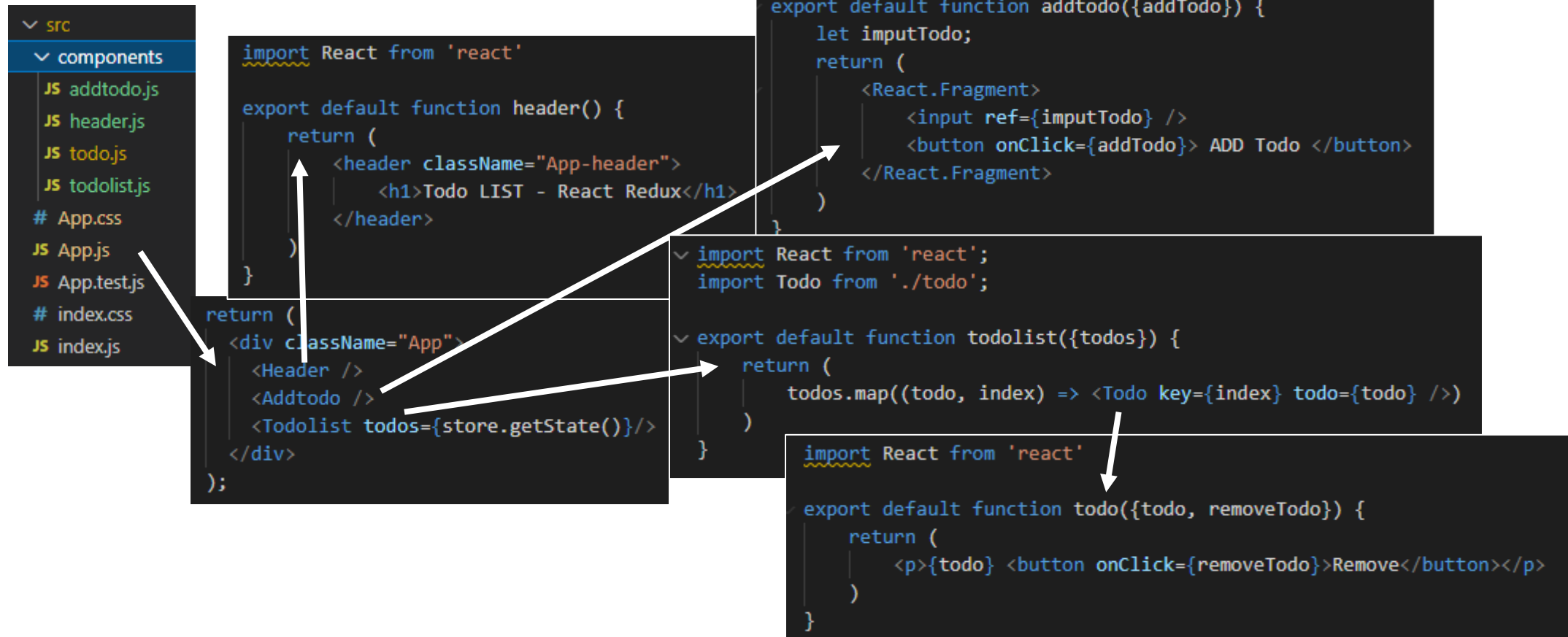
 **Copy**

<https://redux.js.org/basics/usage-with-react>

Per utilizzare **React-Redux** si utilizzano **Componenti di presentazione** e **Componenti contenitore**

	Presentational Components	Container Components
Purpose	How things look (markup, styles)	How things work (data fetching, state updates)
Aware of Redux	No	Yes
To read data	Read data from props	Subscribe to Redux state
To change data	Invoke callbacks from props	Dispatch Redux actions
Are written	By hand	Usually generated by React Redux

Organizziamo la nostra To Do List in Componenti



The diagram illustrates the organization of a To Do List application into components. It shows a file explorer on the left and four code snippets representing different components.

File Explorer Structure:

- src
 - components
 - addtodo.js
 - header.js
 - todo.js
 - todolist.js
 - App.css
 - App.js
 - App.test.js
 - index.css
 - index.js

Component Code Snippets:

```
import React from 'react'

export default function header() {
  return (
    <header className="App-header">
      <h1>Todo LIST - React Redux</h1>
    </header>
  )
}
```

```
import React from 'react'

export default function addtodo({addTodo}) {
  let inputTodo;
  return (
    <React.Fragment>
      <input ref={inputTodo} />
      <button onClick={addTodo}> ADD Todo </button>
    </React.Fragment>
  )
}
```

```
import React from 'react';
import Todo from './todo';

export default function todolist({todos}) {
  return (
    todos.map((todo, index) => <Todo key={index} todo={todo} />)
  )
}
```

```
import React from 'react'

export default function todo({todo, removeTodo}) {
  return (
    <p>{todo} <button onClick={removeTodo}>Remove</button></p>
  )
}
```

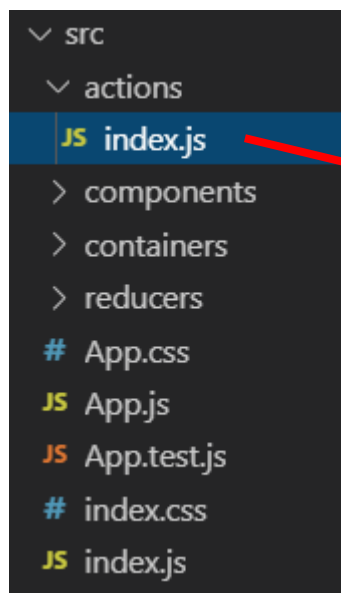
App.js Implementation:

```
import React from 'react'
import { Provider } from 'react-redux'
import { createStore } from 'redux'
import rootReducer from './reducers'
import App from './App'

const store = createStore(rootReducer)

export default function App() {
  return (
    <Provider store={store}>
      <App />
    </Provider>
  )
}
```

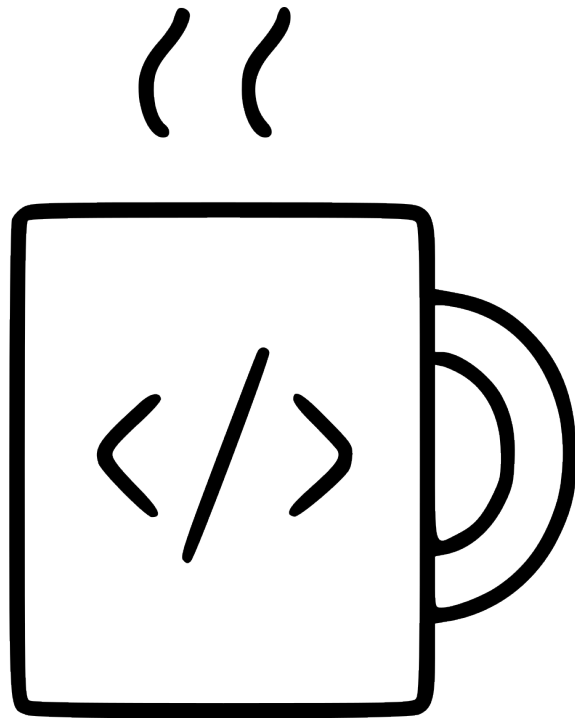
Organizziamo la nostra To Do List in Componets, Container, Reducers e Action



```
export const addTodo = (todo) => {
  return {
    type: 'ADD_TODO',
    todo: todo
  }
}

export const removeTodo = (id) => {
  return {
    type: 'REMOVE_TODO',
    id: id
  }
}
```

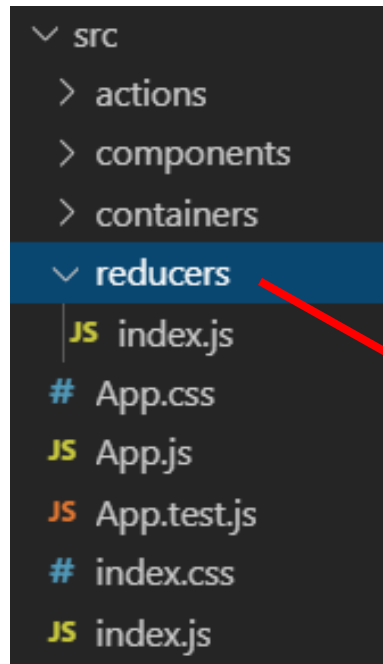
Visto che il dispatch non sarà più manuale, sposto e modifico i metodi **addTodo** e **removeTodo** da App.js in una sezione actions



PAUSA

Ci vediamo alle ore 11.15

Organizziamo la nostra To Do List in Componets, Container, Reducers e Action



Sposto e modifico il Reducer

```
export default function storeReducer(state = [], actions) {  
  switch(actions.type) {  
    case 'ADD_TODO':  
      //alert(actions.todo);  
      return [actions.todo, ...state]  
    case 'REMOVE_TODO':  
      //alert(actions.index);  
      return [...state.slice(0, actions.index), ...state.slice(actions.index+1)]  
    default:  
      return [...state];  
  }  
}
```

Organizziamo la nostra To Do List in Componets, Container, Reducers e Action

Sposto lo store nella root del nostro progetto.

```
✓ src
  > actions
  > components
  > containers
  > reducers
  # App.css
  JS App.js
  JS App.test.js
  # index.css
  JS index.js
```

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';

import {createStore} from 'redux';
import storeReducer from './reducers/index'

const todos = ['Lavorare', 'Studiare React', 'Giocare a calcetto'];
const store = createStore(storeReducer , todos); //il secondo param

ReactDOM.render(<App />, document.getElementById('root'));
```

```
import './App.css';
import Todolist from './components/todolist';
import Header from './components/header';
import Addtodo from './components/addtodo';
```

```
function App() {
  return (
    <div className="App">
      <Header />
      <Addtodo />
      <Todolist todos={[]} />
    </div>
  );
}

export default App;
```

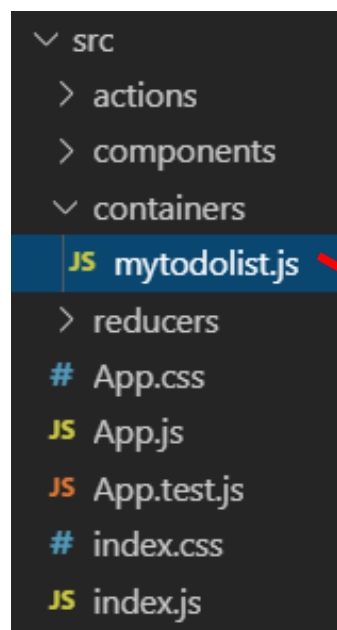

Organizziamo la nostra To Do List in Componets, Container, Reducers e Action

Per iniettare lo **store** nella nostra applicazione utilizzeremo un **Provider** di react-redux. Qualunque componente interno al provider può fare il subscribe allo store. Evitiamo così il passaggio infinito di props da componente padre a componente figlio

```
import {createStore} from 'redux';  
import storeReducer from './reducers/index'  
import {Provider} from 'react-redux';  
  
const todos = ['Lavorare', 'Studiare React', 'Giocare a calcetto'];  
const store = createStore(storeReducer , todos); //il secondo paramentro todos è lo stato iniz  
  
ReactDOM.render(<Provider store={store}><App /></Provider>, document.getElementById('root'));
```

React - Redux

Creiamo ora un componente **Container** che colleghi lo **store** con la nostra **Todolist**. Tramite connect facciamo il subscribe automatico tra il componente avvolto (todolist) e lo store



```
import { connect } from 'react-redux';
import Todolist from '../components/todolist';

const mapStateToProps = (state) => {
  return {todos: [...state]};
}

const myConnect = connect(mapStateToProps); //connect ritorna una funzione
const myToDoList = myConnect(Todolist);

//const myToDoList = connect(mapStateToProps)(Todolist); //Equivalente

export default myToDoList;
```

```
function App() {
  return (
    <div className="App">
      <Header />
      <Addtodo />
      { /* <Todolist todos={[]} /> */}
      <MyToDoList />
    </div>
  );
}

export default App;
```

connect accetta fino a 2 parametri:

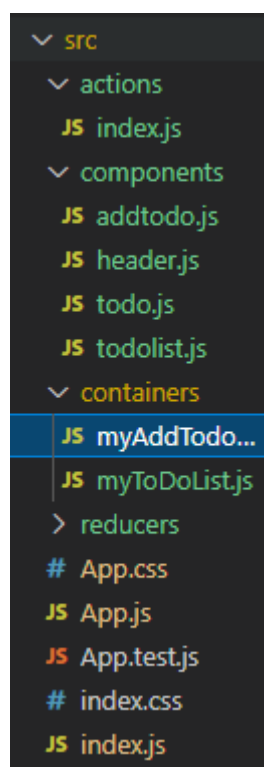
1) **mapStateToProps** -> una funzione che torna un oggetto; le proprietà che inserite in quest'oggetto saranno i valori che leggerete dallo store; questi valori arrivano nelle props

2) **mapDispatchToProps** -> una funzione che torna un oggetto: le proprietà che inserite in quest'oggetto saranno le azioni che potrete dispatchare da questo componente

```
const mapStateToProps = (state) => ({
  cartContent: state.cart.content,
  // ora cartContent è accessibile nel componente tramite this.props.cartContent
})
const mapDispatchToProps = (dispatch) => ({
  // mapDispatchToProps viene riempito di METODI
  addRandomBookToCart: (randomBook) => {
    dispatch({
      type: ADD_TO_CART,
      payload: randomBook,
    })
  },
})
```

```
export default connect(mapStateToProps, mapDispatchToProps)(ClassComponent)
```

Organizziamo la nostra To Do List in Componets, Container, Reducers e Action



```
import React from 'react';
import {addTodo} from '../actions/index';

export default function addtodoComponent({dispatch}){
  let todoInput;
  return (
    <React.Fragment>
      <input ref={node => {todoInput = node}} />
      <button onClick = {
        () => {dispatch(addTodo(todoInput.value))
          todoInput.value = '';}
      }>Add</button>
    </React.Fragment>
  );
}
```

```
import { connect } from 'react-redux';
import Addtodo from '../components/addtodo';

const myConnect = connect();
const myAddTodo = myConnect(Addtodo);

export default myAddTodo
```

Creiamo un componente **Container** che gestirà l'inserimento di dati

```
function App() {
  return (
    <div className="App">
      <Header />
      { /* <Addtodo /> */ }
      <MyAddTodo />
      { /* <Todolist todos={[]} /> */ }
      <MyToDoList />
    </div>
  );
}
```

Organizziamo la nostra To Do List in Componets, Container, Reducers e Action

```
export default function addtodoComponent({addNew}){
  let todoInput;
  return (
    <React.Fragment>
    <input ref={node => {todoInput = node}} />
    <button onClick = {
      () => {addNew(todoInput.value)
              todoInput.value = '';}
    }>Add</button>
    </React.Fragment>
  );
}
```

Separiamo la logica dal componente

```
import {connect} from 'react-redux';
import AddtodoComponent from '../components/addtodo';
import { addTodo } from '../actions';

/*const mapDispatchToProps = (dispatch) => {
  return {
    addNew: (todo) => {
      dispatch(addTodo(todo))
    }
  }
}

const myConnect = connect(null, mapDispatchToProps);
*/

const methods = {addNew: addTodo}
const myConnect = connect(null, methods);
const MyAddTodo = myConnect(AddtodoComponent);

export default MyAddTodo;
```


Organizziamo la nostra To Do List in Componets, Container, Reducers e Action

```
import {connect} from 'react-redux';
import TodoList from '../components/todolist';
import {removeTodo} from '../actions/index';

const mapStateToProps = (state, ownProps) => {
  console.log(state);
  return {
    todos: [...state]
  }
}

const myConnect = connect(mapStateToProps, {removeTodo});
const MyTodoList = myConnect(TodoList);

//const MyTodoList = connect(mapStateToProps)(TodoList) //

export default MyTodoList;
```

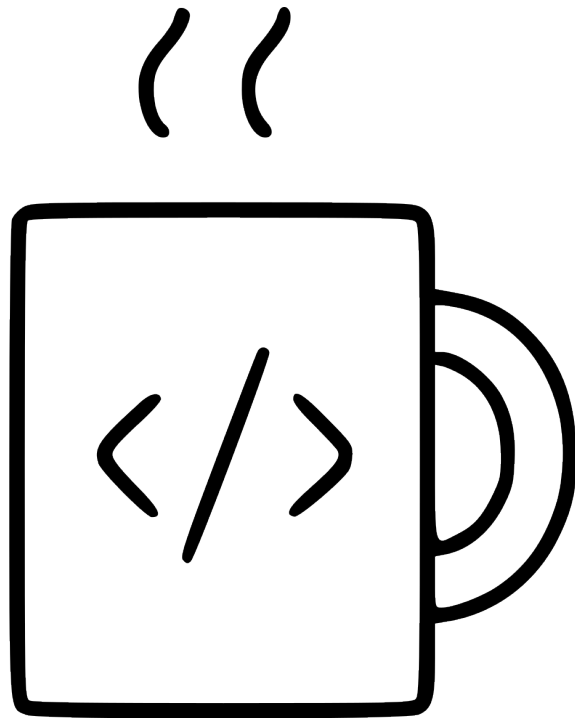
```
import React from 'react';
import Todo from './todo';

export default function todolist({todos, removeTodo}){
  return todos.map((t,id) => <Todo id={id} key={id} onClick={removeTodo} todo={t} />)
}
```

```
import React from 'react';

export default function todo({todo, onClick, id}){
  return <p> {todo} <button onClick={() => onClick(id)}>X</button></p>
}
```

Gestiamo la cancellazione dai dati sfruttando MyTodoList



PAUSA

Ci vediamo alle ore 14.00

Supponiamo ora di voler recuperare i dati da un server, effettuando chiamate API.

Qual è il posto in cui implementare funzioni asincrone in Redux?

Il **reducer** deve sempre essere una **funzione pura** quindi, dato un input, deve restituire sempre lo stesso output. Non va utilizzato, dunque, per chiamate API o per produrre effetti collaterali come chiamate AJAX.

React - Redux

Le **actions** sono oggetti semplici, ma possiamo provare ad inserirle in un action creator che è una funzione. Grazie a Redux Thunk, un middleware per Redux puoi restituire funzioni da action creators e puoi eseguire logica asincrona all'interno delle tue actions

```
npm i redux-thunk
```

Inside index.js

```
import thunk from "redux-thunk"
import { createStore, applyMiddleware } from 'redux';
...createStore(rootReducers, applyMiddleware(thunk));
```

```
//Async Action
export function getTodo() {
  return function(dispatch) {
    return fetch("https://jsonplaceholder.typicode.com/todos")
      .then(response => {
        if (!response.ok) throw Error(response.statusText);
        return response.json();
      })
      .then(json => {
        return dispatch({ type: "TODO_LOADED", payload: json });
      });
  };
}
```

Redux-thunk permette di inserire un middleware di mezzo che aiuta a gestire le operazioni asincrone o controlli.

npm i redux-thunk

Inside index.js

```
import thunk from "redux-thunk"
import { createStore, applyMiddleware } from 'redux';

...createStore(rootReducers, applyMiddleware(thunk));
```

```
export const addTodo = (todo) => {
  return (dispatch, getState) => {
    // getState ritorna lo stato contenuto nello store
    console.log(getState());
    if(getState().findIndex(t => t.todo === todo) < 0){
      dispatch({ type: 'ADD_TODO', todo: todo})
    }
  }
}
```


React - Redux

Fatto ciò, possiamo aggiungere la nuova action al nostro reducer:

```
export default function storeReducer(state = {}, action) {  
  //console.log(action);  
  switch(action.type) {  
    case 'ADD_TODO':  
      return {  
        ...state,  
        todos: [...state.todos, {todo: action.todo, resolve: ''}]  
      }  
    case 'REMOVE_TODO':  
      return {  
        ...state,  
        todos: [...state.todos.slice(0, action.id), ...state.todos.slice(action.id+1)]  
      }  
    case 'RESOLVE_TODO':  
      state.todos[action.id].resolve = state.todos[action.id].resolve === 'resolve' ? '' : 'resolve';  
      return {  
        ...state,  
        todos: state.todos  
      }  
    case 'TODO_LOADED':  
      return Object.assign({}, state, {  
        remoteTodos: state.remoteTodos.concat(action.payload)  
      });  
    default:  
      return state;  
  }  
}
```

React - Redux

Infine, creiamo il nostro componente con il suo container per visualizzare i nostri todos remoti:

```
const mapStateToProps = state => {
  return {todos: state.remoteTodos.slice(0, 10)}
}

const FetchTodolist = (props) => {
  useEffect(() => {
    props.getTodo();
  }, []);
  return (
    <Container>
      <ListGroup>
        {props.todos.map((todo,i) => <ListGroup.Item>{todo.title}</ListGroup.Item>)}
      </ListGroup>
    </Container>
  )
}

export default connect(mapStateToProps, { getTodo })(FetchTodolist);
```



shaping the skills of tomorrow

challengenetwork.it

