



# Full-stack Application Development

Redux – predictable state container for JavaScript apps

# Where to Find The Code and Materials?

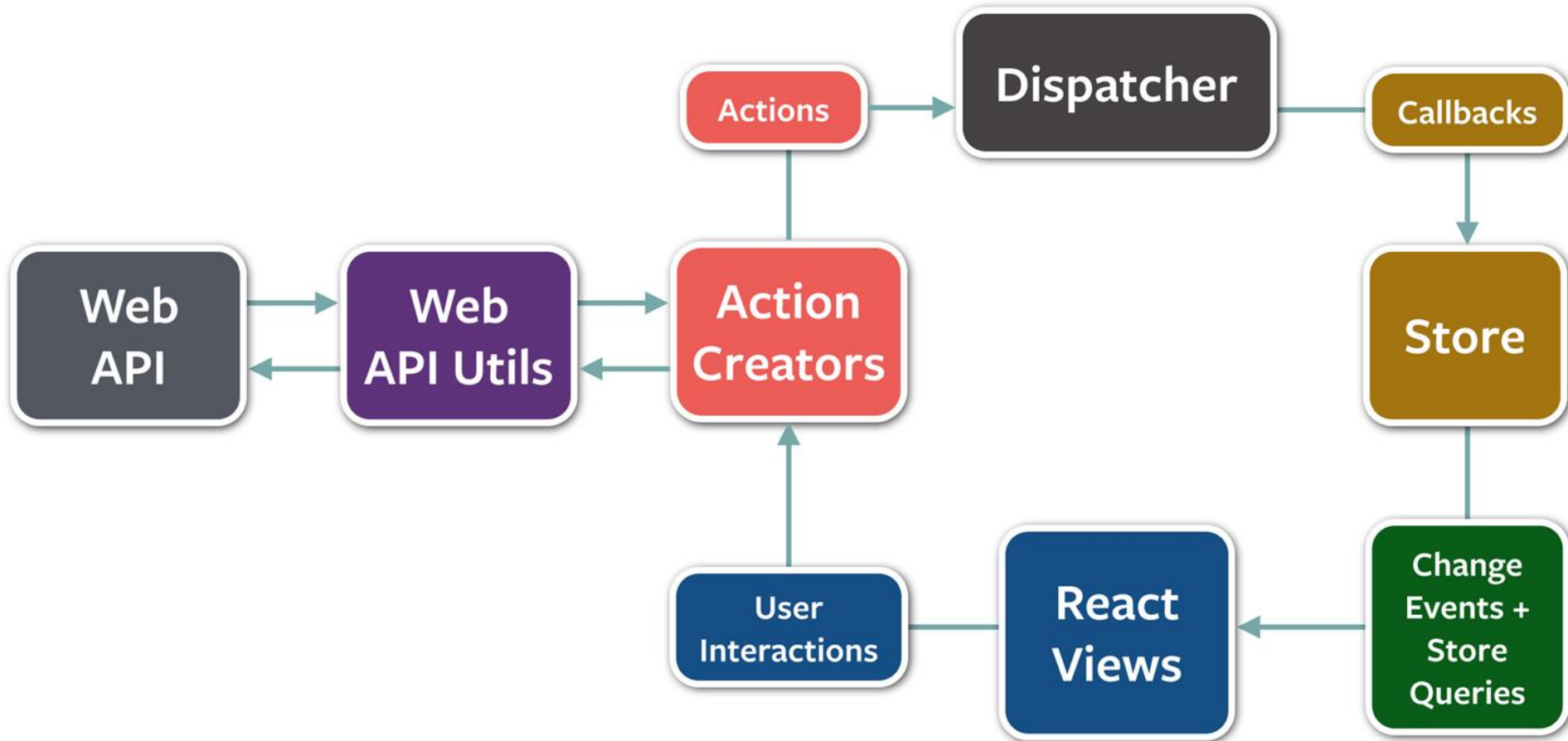
<https://github.com/iproduct/fullstack-typescript-react>



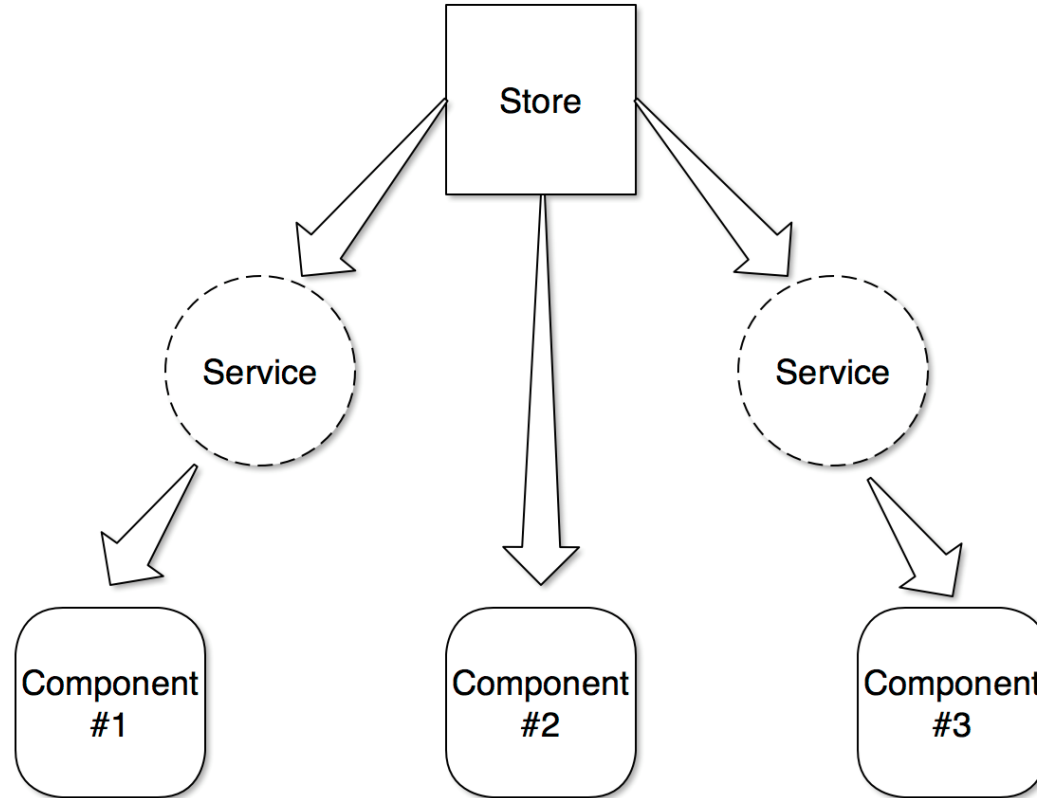
# Agenda

1. Flux and Redux design patterns
2. Main components: store, actions, action creators, reducers, selectors
3. Redux Recommended (Basic) Project Structure
4. Bootstrapping the Redux App
5. Redux Action Creators
6. Redux Reducers
7. Redux Root Reducer
8. Redux Containers and *connect* decorator
9. Using Redux Router Redux and Redux Thunk
10. Advanced Redux using Middleware Libraries

# Flux Design Pattern



# Redux Design Pattern



**Linear flow:** Dispatch → Reducers → New State → Store

# Redux

- Streamlined state management for **React.js** applications, inspired by **Redux**
- **State** is a single immutable data structure
- **Actions** describe state changes
- Pure functions called **reducers** take the **previous state** and the **next action** to compute the **new state**
- State is kept within single **Store**, and accessed through sub-state **selectors**, or as **Observable** of state changes
- Components are by default performance optimized using the **shouldComponentUpdate()** → **performant change detection**

# Redux Recommended (Basic) Project Structure

- **actions** – action creator factory functions (design pattern Command)
- **assets** – static assets (css images, fonts, etc.) folder
- **components** – simple (dumb) react components – pure render
- **container** – Redux Store aware (smart) component wrappers
- **reducers** – the only way to advance state:

`function(OldStoreState, Action) => NewStoreState // = Rx scan()`

- **index.js** – bootstraps app providing access to Store for all containers (smart components) using React context

# Bootstrapping the Redux App – index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Provider } from 'react-redux';
import { createStore } from 'redux';
import rootReducer from './reducers';
import { FilteredTodoApp } from './containers/filtered-todo-app';

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

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

← Redux store provider

← Top level container component



# Redux Action Creators – /actions/index.js

```
let nextTodoId = 0;
export const addTodo = (text) => ({
  type: 'ADD_TODO',
  id: nextTodoId++,
  text
});
export const setVisibilityFilter = (filter) => ({
  type: 'SET_VISIBILITY_FILTER',
  filter
});
export const changeStatus = (id, status) => ({
  type: 'CHANGE_STATUS',
  id,
  status
});
...
```

← Action Type

} ← Action Payload

# Redux Reducers – /reducers/todo.js

```
const todoReducer = (state = {}, action) => {  
  switch (action.type) {  
    case 'ADD_TODO':  
      return {  
        id: action.id,  
        text: action.text,  
        status: 'active'  
      };  
    case 'CHANGE_STATUS':  
      if (state.id !== action.id) {  
        return state;  
      }  
      return Object.assign({}, state, { status: action.status });  
    default:  
      return state;  
  }  
};
```

# Redux Reducers – /reducers/todos.js

```
const todosReducer = (state = [], action) => {  
  switch (action.type) {  
    case 'ADD_TODO':  
      return [  
        ...state,  
        todoReducer(undefined, action)  
      ];  
    case 'CHANGE_STATUS':  
      return state.map( todo => todoReducer(todo, action) );  
    case 'DELETE_TODOS':  
      return state.filter(todo =>  
        todo.status !== action.status  
      );  
    default:  
      return state;  
  }  
};
```

# Redux Root Reducer – /reducers/index.js

```
import { combineReducers } from 'redux';  
import todos from './todos';  
import visibilityFilter from './visibilityFilter';  
  
const rootReducer = combineReducers({  
  todos,  
  visibilityFilter  
});  
  
export default rootReducer;
```

# Redux Containers – /containers/visible-todo-list.js

```
const getVisibleTodos = (todos, filter) => todos.filter(
  todo => filter === 'all' ? true: todo.status === filter);
const mapStateToProps = (state) => ({
  todos: getVisibleTodos(state.todos, state.visibilityFilter)
});
const mapDispatchToProps = (dispatch) => ({
  onCompleted: (id) => {
    dispatch(changeStatus(id, 'completed'));
  },
  onCancel: (id) => {
    dispatch(changeStatus(id, 'canceled'));
  }
});
const VisibleTodoList = connect(mapStateToProps, mapDispatchToProps) (TodoList);
export default VisibleTodoList;
```

# Redux App using ES7 @connect Decorator

```
const getVisibleTodos = (todos, filter) => todos.filter(
  todo => filter === 'all' ? true: todo.status === filter);
const mapStateToProps = (state) => ({
  todos: getVisibleTodos(state.todos, state.visibilityFilter)
});
const mapDispatchToProps = (dispatch) => ({
  onCompleted: (id) => {
    dispatch(changeStatus(id, 'completed'));
  },
  onCancel: (id) => {
    dispatch(changeStatus(id, 'canceled'));
  }
});
@connect(mapStateToProps, mapDispatchToProps)
export default class TodoList extends React.Component {
  constructor(props) { ...
```

# Using Redux Router Redux and Redux Thunk (1)

- Idea: `history + store (redux) → react-router-redux → enhanced history → react-router`

```
import { compose, createStore, combineReducers, applyMiddleware } from 'redux';
import { Provider } from 'react-redux';
import createHistory from 'history/createBrowserHistory';
import { ConnectedRouter, routerReducer, routerMiddleware, push } from
  'react-router-redux';          // React Router to Redux integration
import thunk from 'redux-thunk'; // Allows using thunks = async actions
import reducers from './reducers'; // your reducers here
const history = createHistory(); // use browser History API
// middleware for intercepting & dispatching navigation & async actions
const middleware = [routerMiddleware(history), thunk];
// Enable Redux Devtools
const composeEnhancers = window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ || compose;
```

# Using Redux Router Redux and Redux Thunk (2)

```
const store = createStore(  
  combineReducers({ ...reducers,  
    router: routerReducer // Add the reducer to store on `router` key  
  }),  
  /* preloadedState, */  
  composeEnhancers( applyMiddleware(...middleware) ));  
store.dispatch(push('/repos/react/redux')); //dispatch navigation action  
ReactDOM.render(  
  <Provider store={store}> //ConnectedRouter use store from the Provider  
    <ConnectedRouter history={history}>  
      <App />  
    </ConnectedRouter>  
  </Provider>,  
  document.getElementById('root')  
)
```



# Redux Thunk Async Actions - /actions/counter.js

```
export function increment(x) {  
  return { type: INCREMENT, amount: x }  
}
```

```
export function incrementAsync(x) {  
  return dispatch => //Can invoke sync or async actions with `dispatch`  
    setTimeout(() => { dispatch(increment(x)); }, 2000);  
}
```

```
export function incrementIfOdd(x) {  
  return (dispatch, getState) => {  
    const { counter } = getState();  
    if (counter.number % 2 === 0) return;  
    dispatch(increment(x)); //Can invoke actions conditionally  
  };  
}
```

# Advanced Redux using Middleware Libraries

- **Normalizr** – [normalizing](#) and [denormalizing](#) data in state, helps to transform nested JSON response structures into a relational DB-like plain entities, referenced by Id in the Redux store.
- **redux-thunk** – in addition to plain actions, **action creators** can now return **Thunks** – callback functions of **dispatch** and **getState** arguments, allowing to handle async operations like [data fetch from REST endpoint](#) and [Promise-like composition](#).
- **redux-promise/ redux-promise-middleware** – thunk alternatives
- **redux-observable** – really powerfull reactive transforms of async action events as RxJS Observables, called **Epics**:

**(action\$:Observable<Action>, store:Store) => Observable<Action>**

- **redux-saga** - a library that aims to make application side effects (i.e. asynchronous things like data fetching and impure things like accessing the browser cache) easier to manage, more efficient to execute, easy to test, and better at handling failures. Uses [generator functions](#) instead of promises.

# Thank's for Your Attention!



Trayan Iliev

IPT – Intellectual Products & Technologies

<http://iproduct.org/>

<http://robolearn.org/>

<https://github.com/iproduct>

<https://twitter.com/trayaniliev>

<https://www.facebook.com/IPT.EACAD>