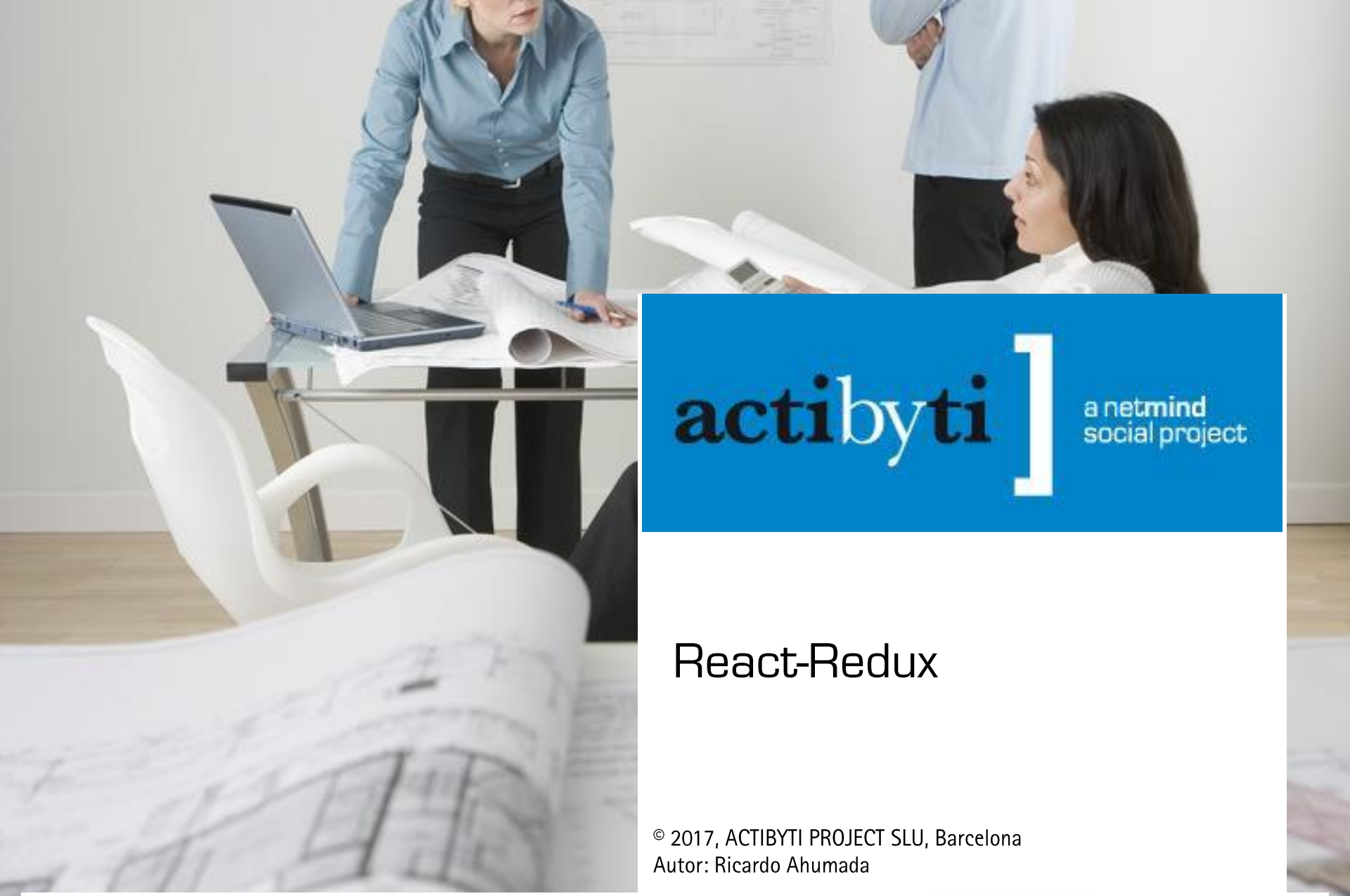


actibyti] a netmind social project

React-Redux

Autor: Ricardo Ahumada

GOBIERNO DE ESPAÑA
MINISTERIO DE ENERGÍA, TURISMO Y AGENDA DIGITAL
red.es
ESTRATEGIA DE EMPRENDIMIENTO Y EMPLEO JOVEN garantía juvenil
Agenda Digital para España
UNIÓN EUROPEA
Fondo Social Europeo
"El FSE invierte en tu futuro"

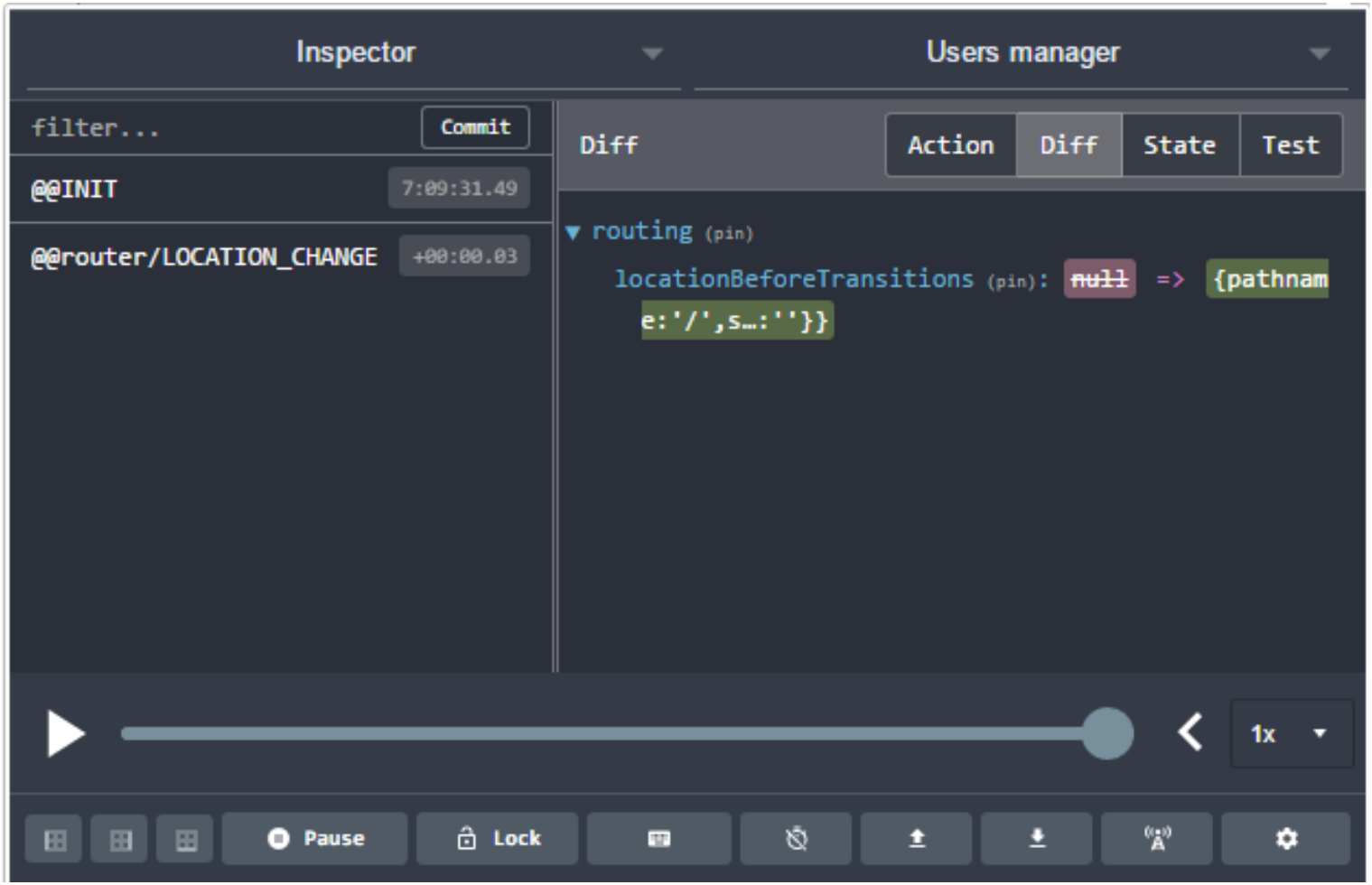
1 Configurando Redux

Librerías

- Será necesario añadir las librerías de Redux a nuestro proyecto (además de las que necesita React)
  - react-redux: Conecta react con redux
    - https://github.com/reactjs/react-redux/tree/master/docs
  - redux: La librería redux propiamente dicha
    - http://redux.js.org/, http://redux.js.org/docs/api/
  - redux-thunk: Si queremos lanzar acciones asíncronas
    - https://github.com/gaearon/redux-thunk

Herramientas: Depuración

- ***Redux DevTools***:
  - Es una extensión para Chrome para el desarrollo Redux.



2 Proceso Flux

Escenario de gestión de estado

- En una SPA en lugar de almacenar datos en el DOM o en objetos aleatorios, existe un conjunto de modelos en memoria que representan todo el estado / datos en la aplicación.
- Una vista que puede actualizar un modelo, y este modelo puede actualizar otro modelo, una vista de calidad u otra vista, que podría tener el mismo modelo.
- Posibles efectos:
  - Descontrol de la aplicación
  - Relentización y bajo performance
- En estas situaciones se necesita una “única fuente de verdad”
- Queremos que las vistas reflejen este estado y que acciones sobre ella actualicen dicho estado a su vez

Integración de herramientas

- React es una librería que provee las funcionalidades necesarias para generar la vista del frontend una aplicación. No provee de la parte del Controlador y Modelo de la misma.
- Si la aplicación tiene cierto grado de complejidad y necesita compartir estado es necesario integrar react con otros frameworks que provean la gestión de cambios de estado y permitan implementar de manera segura el intercambio del mismo.
- Dado que React promueve el flujo de en una sola dirección del estado es recomendable implementar el Patrón Flux

El patrón Flux

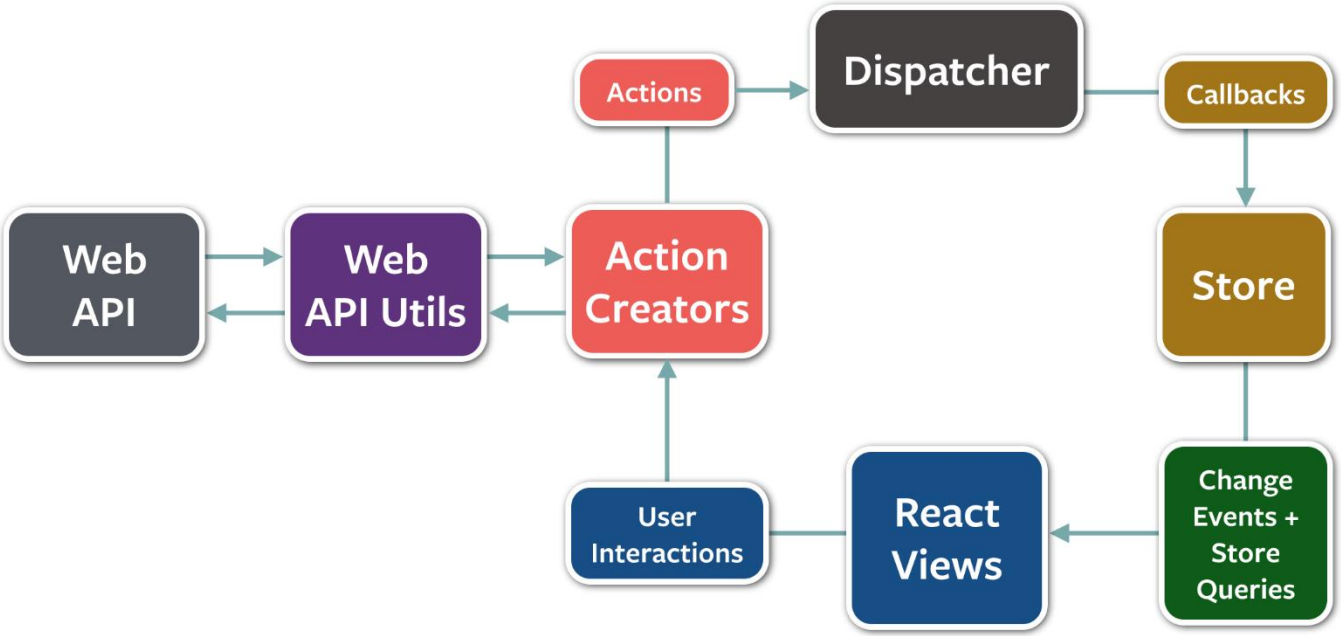
- Flux es una arquitectura que usa un flujo de datos unidireccional a través de eventos y listeners específicos.
- No es un framework, sino más bien un patrón.
- No es lo mismo que el patrón MVC. Los controladores no existen en Flux.

Consiste de 4 componentes:

- **Dispatchers**: Un singleton que actúa como registro de callbacks y opera como hub central del flujo de datos.
  - Cuando un dato llega, el dispatcher llama a la función de callback y propaga el estado.
  - El proceso de llamada se inicia con un **dispatch()**
- **Almacenes (Stores)**: Una colecciones de datos y lógica de negocio expuesta como Singleton. Contiene los modelos.
- **Vistas**: se encarga de mostrar los datos y presentar las funcionalidades de una aplicación.
- **Acciones**: Son objetos que encapsulan los datos que modificarán los Stores mediante un tipo de acción.
- Para implementar Flux se pueden usar distintas librerías

Proceso Flux

- La Vista emite acciones (ej. acciones de usuarios)
- Se crean acciones que encapsulan lo que ha cambiado
- El Dispatcher responde a las acciones emitidas llamando callbacks
- El Store emite un evento de cambio
- La vista responde al evento de cambio



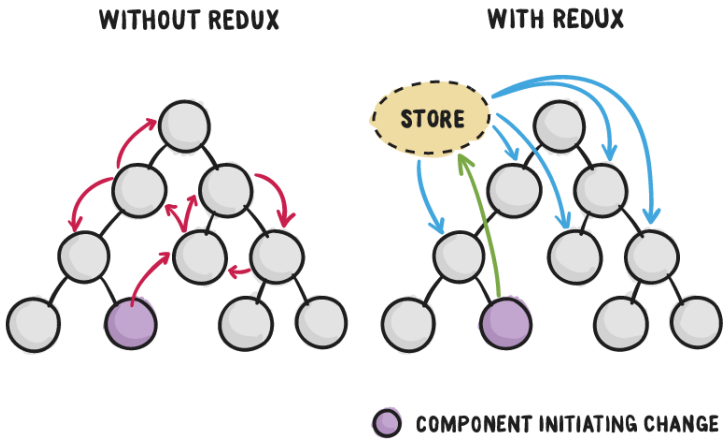
Posibles librerías para implementar Flux

- No existe una versión oficial de librería Flux, por tanto, escoger entre las alternativas existentes se debe realizar de manera muy cuidadosa:
  - Flummox
  - Alt
  - Fluxxor
  - Flux This
  - MartyJS
  - McFly
  - Fluxible
  - Delorean
  - Lux
  - Reflux
  - OmniscientJS
  - Fluxy
  - Material Flux
- Actualmente Redux se ha posicionado como una alternativa simple y eficiente para implementar el modelo **Flux**.

3 Redux

Redux

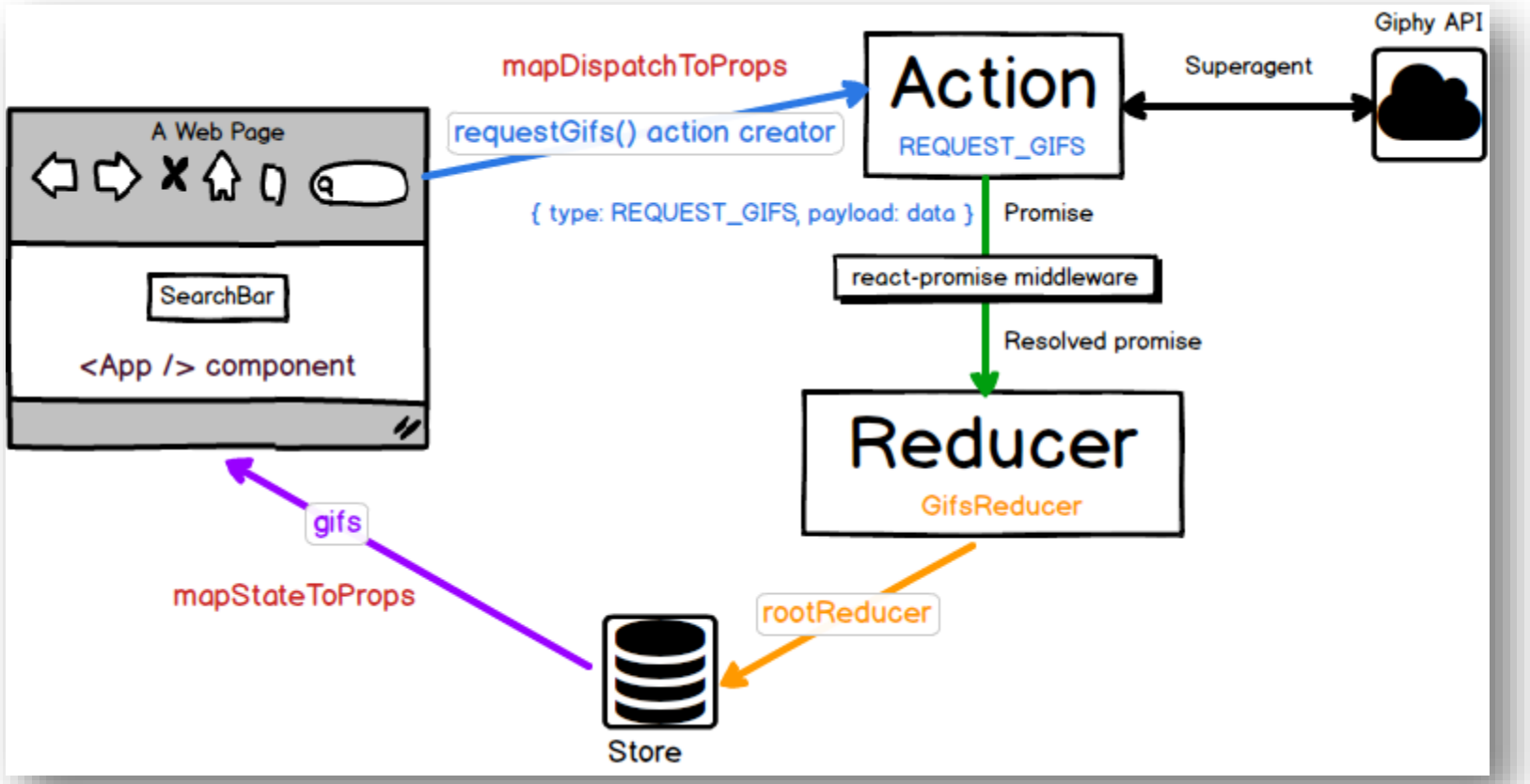
- Redux (http://redux.js.org/) es un contenedor de estado predecible para aplicaciones JS
- Es una implementación ligera de flux (2KB)
- Principales diferencias con otras implementaciones Flux:
  - No hay "**dispatchers**" discretos. Directamente la "store" está a la escucha de acciones y usa los "**reducers**" para devolver un estado nuevo.
  - Mantiene todo el estado de la aplicación en un único sitio: la **store**.
  - El estado de la aplicación es inmutable.



Redux - Componentes

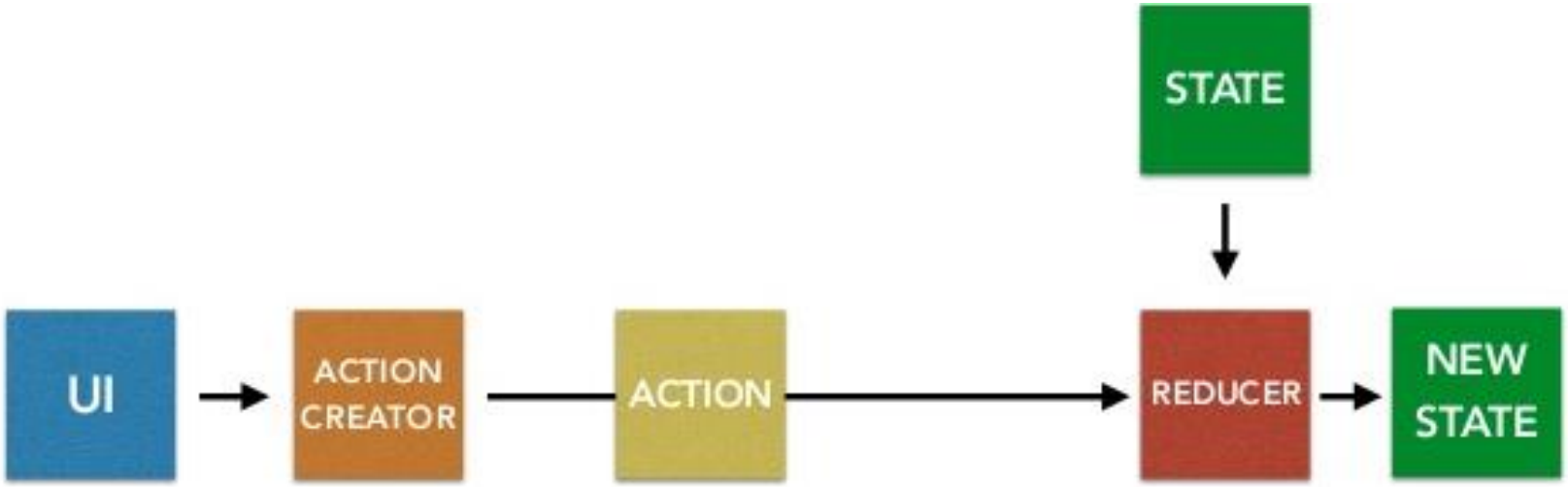
- Store: objeto donde se almacenan los modelos y sus estados
  - Se declarará en un punto de entrada
- ActionCreators: Factory de **Actions** (objetos indican que algo ha sucedido)
- Reducers: Especifica cómo cambia el estado de la aplicación
- Componentes React:
  - Componentes de presentación
  - Componentes contenedores
  - Componentes funcionales

Redux – Componentes (II)



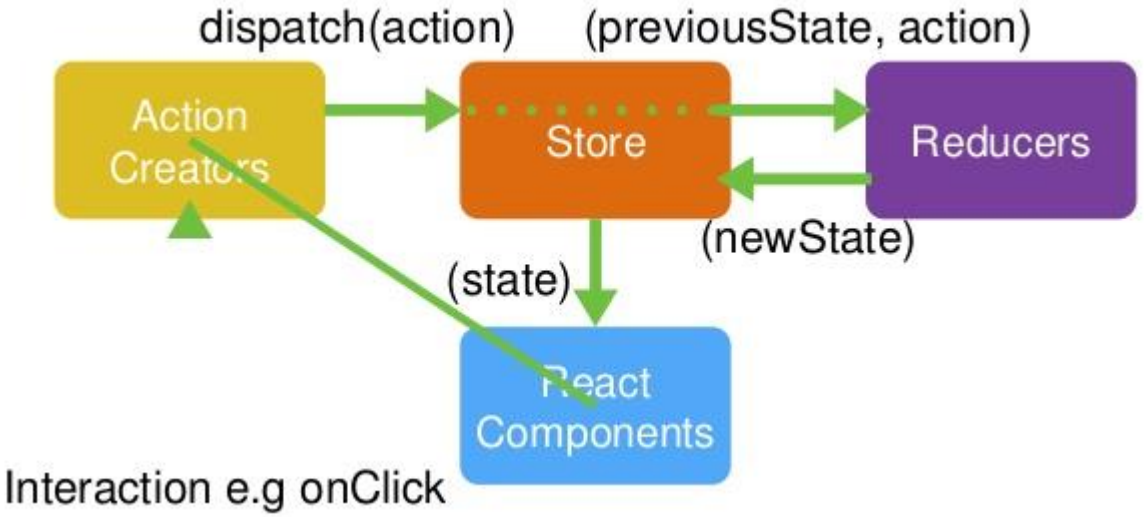
Redux - Proceso

- Toda interacción en la UI genera una acción que se envía a la store
- Se llama a un action creator → esta acción le llega a un reducer
  - Los reducers deben ser diseñados como funciones puras
- El reducer toma el estado actual, la acción y genera un nuevo estado.
- La store actualiza su estado con el nuevo estado generado por el reducer



Redux – Flujo de datos

- Se hace dispatch de una acción: store.dispatch(action)
- La Store llama a la función reducer asignada (en el gluing de reducers a store). Una por cada modelo.
- El root reducer puede combinar múltiples reducers para generar un solo árbol de estado.
- El store guarda el nuevo estado emitido por el root reducer
- El Componente sincronizado con el estado de la store se actualiza



Redux - Ejemplos

- El creador de Acciones

```
export function addUser(name, email){
    console.log("Dispatching add User");
    return {
        type:'ADD_USER',
        name,
        email
    }
}
```

- Una Acción

- Un reducer

```
function postUsers(state=[], action){
    switch(action.type){
        case 'ADD_USER':
          // return the new state with the new comment
          return [...state,{
            id: state.length+1,
            name: action.name,
            email: action.email
          }];

        default:
          return state;
    }

    return state;
}
```

Redux – Ejemplos (cont…I)

- El aglutinador de reducers

```
const rootReducer=combineReducers({users
    ,empresas
    ,routing:routerReducer}
    );
```

- Permitirá conectar un reducer combinado con la store
- A su vez asignará a cada reducer su parte del la store correspondiente (un reducer por modelo)
- Finalmente devolverá un árbol de estado actualizado una vez se procese

Redux – Ejemplos (cont...II)

- La store

```
import {createStore,compose} from 'redux';
import {syncHistoryWithStore} from 'react-router-redux';
import {browserHistory} from 'react-router';

// import the route reducer
import rootReducer from './reducers/index';
```

```
const store = createStore(rootReducer,defaultState,enhacers);

export const history = syncHistoryWithStore(browserHistory,store);

export default store
```

- Se crea la store, registrando los reducers (en root reducer)

- Conección React-Redux: Se pasa la store a componentes de routing como prop

```
const router = (
    <Provider store={store} url="api/users.json">
        <Router history={history} >
            <Route path="/" component={App}>
                <IndexRoute component={UserBox}></IndexRoute>
                <Route path="/form" component={UserForm}></Route>
            </Route>
        </Router>
    </Provider>
)
```

Redux – Ejemplos (cont…III)

- Conección dispatchers-models-props

```
function mapStateToProps(state){
    return{
        users: state.users,
        empresas:state.empresas
    }
}
```

- Mapeo del store a props. Cada vez que se actualice la store, se llamará a esta función

```
function mapDispatchToProps(dispatch){
    return bindActionCreators(actionCreator,dispatch);
}
```

- Bind de dipatchers a props usando el factory de acciones

```
const App = connect(mapStateToProps, mapDispatchToProps)(Main);
```

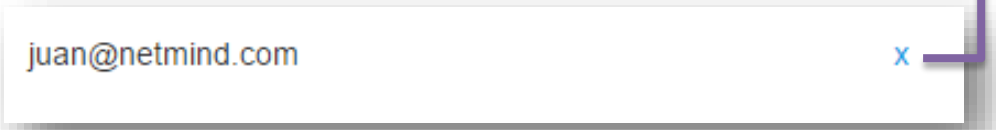
- Sincronizamos la store y dispatchers a los props. Connect deveulve una función, que ejecutamos pasándole como argumento el componente Main

Redux – Ejemplos (cont…IV)

- El dispatch de una acción desde un componente (a través de props)

```
const UserBox = React.createClass({
  deleteUserBoxHandler(id){
    console.log('Delete box user!',id);
    this.props.deleteUser(id,2);
  },
```

- Cada vez que hay un dispatch de acción, todos los reducers son consultados



- En reducer de usuario:

```
Borrando usario... ▶ [Object, Object, Object]
▶ Object {type: "DEL_USER", id: 2, id_empresa: 2}
```

- En reducer de empresa:

```
Borrando datos usuario en empresa...buscar y eliminar las
referencias del  ▶ [Object, Object]
▶ Object {type: "DEL_USER", id: 2, id_empresa: 2}
```

netmind

WeKnowIT

Barcelona

C. Almogàvers, 123
08018 Barcelona
Tel. 93 304.17.20
Fax. 93 304.17.22

Madrid

Plaza Carlos Trías Bertrán, 7
28020 Madrid
Tel. 91 442.77.03
Fax. 91 442.77.07

www.netmind.es

GOBIERNO DE ESPAÑA
MINISTERIO DE ENERGÍA, TURISMO Y AGENDA DIGITAL
red.es
ESTRATEGIA DE EMPRENDIMIENTO Y EMPLEO JOVEN garantía juvenil
Agenda Digital para España
UNIÓN EUROPEA
Fondo Social Europeo
"El FSE invierte en tu futuro"