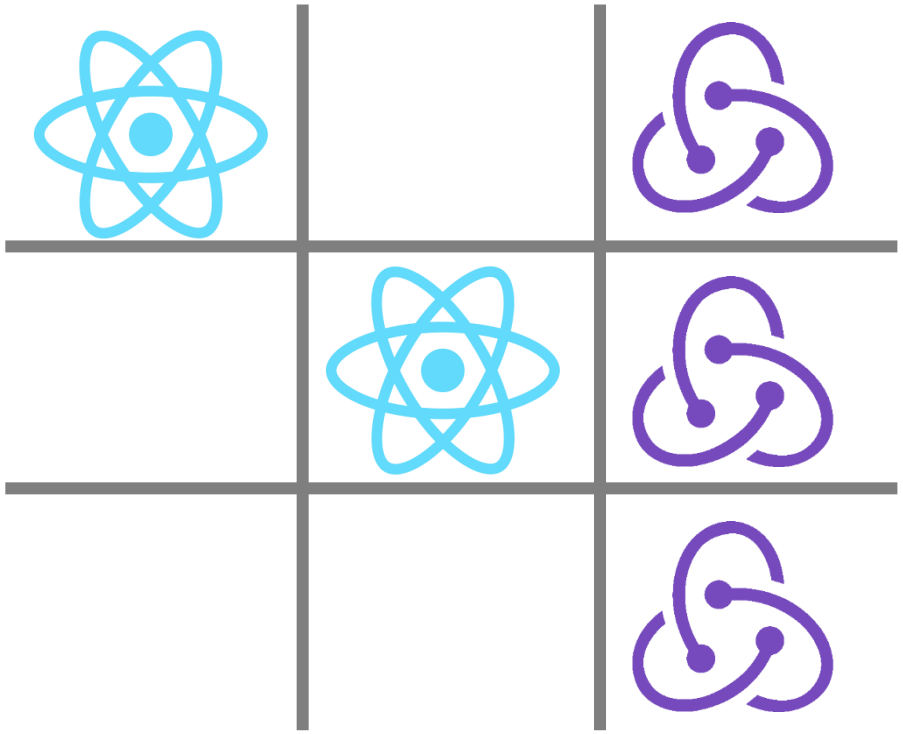


Tres en Raya con Redux

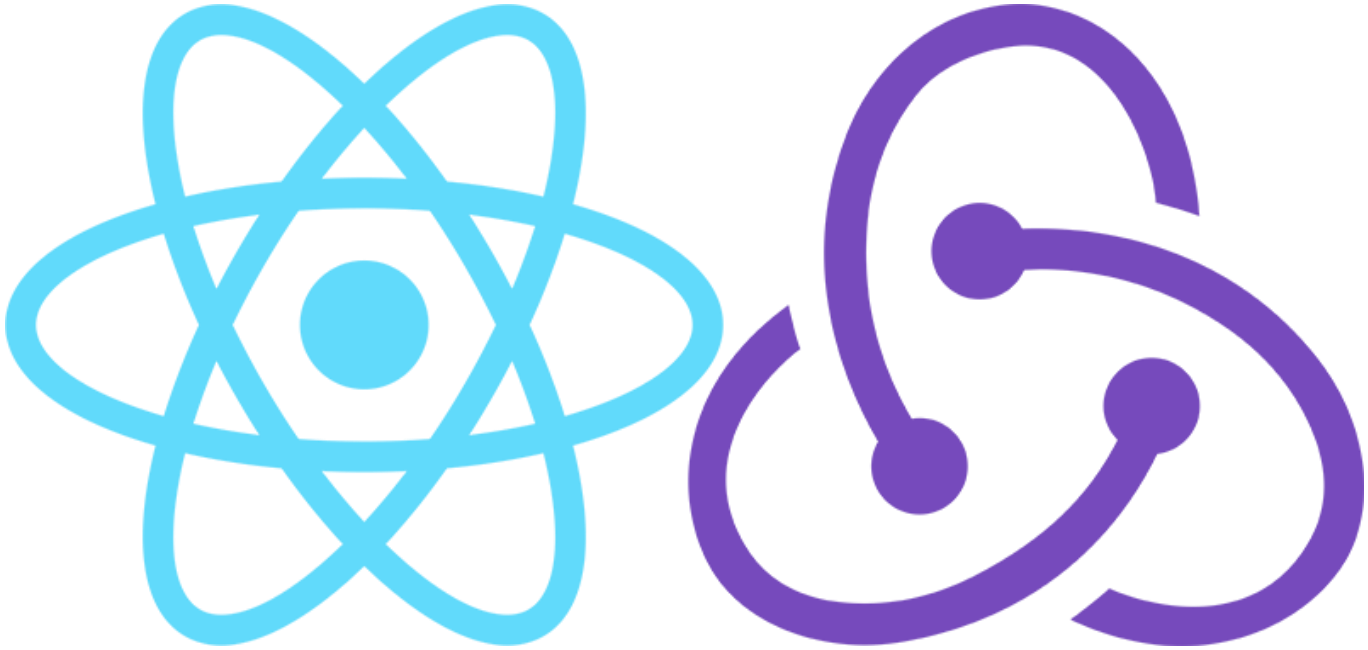
Sonsoles López Pernas

Proceso a seguir

- Instalar dependencias
- Definir el estado de la aplicación
- Definir acciones
- Definir reducers
- Conectar React y Redux

Instalar dependencias

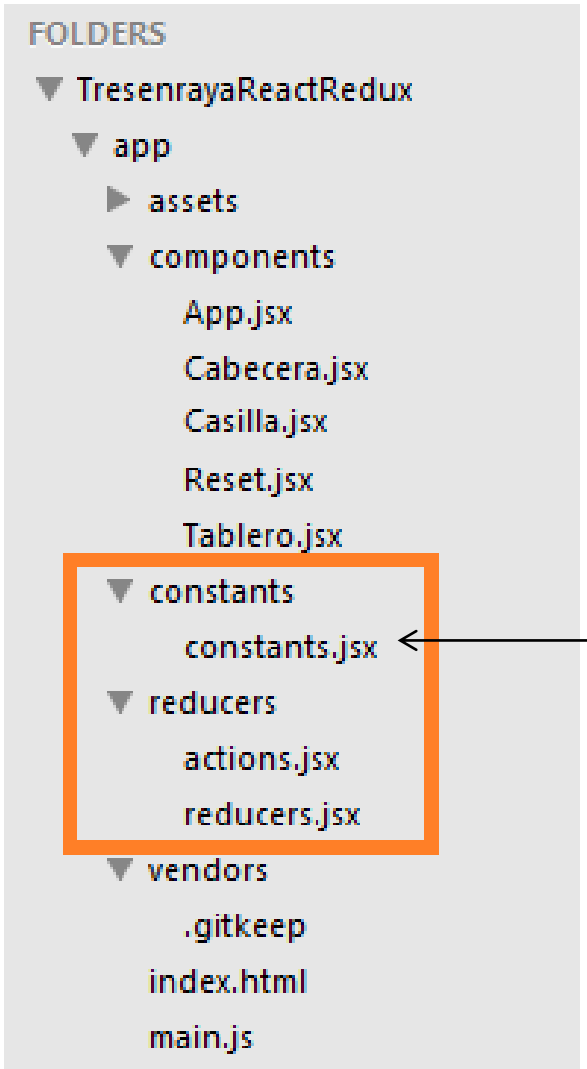
```
npm install --save-dev redux react-redux
```



Defino el estado de la aplicación

- Suele coincidir con el estado de App.jsx en aplicaciones pequeñas.
  - Tablero: Array
  - Turno: String
  - Movimientos: Nº
- Defino el estado inicial (por comodidad pasamos estos valores a un fichero de constantes)
  - Tablero vacío
  - Turno del jugador X (por ejemplo)
  - Cero movimientos

Creo archivos necesarios



constants.jsx

```
export const JUGADORX = "jugador 1 - las X";
export const JUGADOR0 = "jugador 2 - los 0";
export const TABLERO = [
      ['-', '-', '-'],
      ['-', '-', '-'],
      ['-', '-', '-'],
      ]
```

Primera acción – actions.jsx

- Acción **JUGAR_POSICIÓN** : Cada vez que un jugador rellena una casilla.
- ¿Qué información necesita?
  - Qué jugador ha jugado en este turno
  - En qué casilla ha hecho click (x,y)

Primera acción – actions.jsx

```
export function jugarPosicion(x, y, turno) {
  return {
    type : 'JUGAR_POSICION',
    x : x,
    y : y,
    turno: turno
  }
}
```

Reducers

- El estado tiene 3 partes (turno, tablero, movimiento)
- Necesitamos un reducer para cada parte
- Dentro de reducers creamos
  - gameReducer.jsx
  - turnoReducer.jsx
  - movimientosReducer.jsx

- En reducers.jsx unificaremos todas las partes para crear el estado global

El fichero reducers.jsx

```
import { combineReducers } from 'redux';
import gameReducer from './gameReducer';
import turnoReducer from './turnoReducer';
import movimientosReducer from './movimientosReducer';

const GlobalState = combineReducers({
    turno: turnoReducer,
    tablero: gameReducer,
    movimientos: movimientosReducer
});

export default GlobalState;
```

Estructura de un reducer

state no hace referencia al estado completo sino al trozo del estado que le corresponde a este reducer

En action viene toda la información necesaria para modificar el estado, especialmente el type de acción que es.

```
// import {..} …….

function myReducer(state = “ESTADO_POR_DEFECTO”, action) {
  switch (action.type) {
    case 'ACTION_NAME':
      let newState = Object.assign([], state);
       // … Hago modificaciones sobre newState
      return newState; // Devuelvo el estado modificado
    default:
      return state;
  }
}
export default myReducer;
```

gameReducer.jsx

Tenemos que rellenar la casilla especificada con X ó 0 según el jugador.

```
import {JUGADORX, JUGADOR0, TABLERO} from '../constants/constants';

function gameReducer(state = TABLERO, action) {
  switch (action.type) {
    case 'JUGAR_POSICION':
      let nuevoValor = action.turno === JUGADORX ? 'X' : '0';
      let newState = JSON.parse(JSON.stringify(state))
      newState[action.x][action.y] = nuevoValor;
      return newState;
    default:
      return state;
  }
}
export default gameReducer;
```

turnoReducer.jsx

Tenemos que cederle el turno al jugador contrario al que acaba de jugar.

```
import {JUGADORX, JUGADOR0} from '../constants/constants';

function turnoReducer(state = JUGADORX, action) {
  switch (action.type) {
    case 'JUGAR_POSICION':
      return action.turno === JUGADORX ? JUGADOR0 : JUGADORX;
    default:
      return state;
  }
}
export default turnoReducer;
```

movimientosReducer.jsx

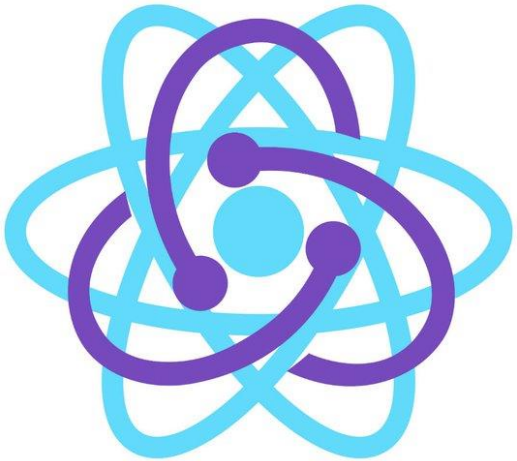
Sólo hay que sumar uno al nº de movimientos anterior

```
function movimientosReducer(state = 0, action) {
  switch (action.type) {
    case 'JUGAR_POSICION':
      return state + 1;
    default:
      return state;
  }
}

export default movimientosReducer;
```

Conectar la aplicación con React

- Componente intermedio entre `main.js` y `App.jsx` => Creamos `ReduxProvider.jsx` en `app/components`
- Tenemos que modificar `App.jsx` y `main.js`



ReduxProvider.jsx

```
import { Provider } from 'react-redux';
import GlobalState from './../reducers/reducers';
import { createStore, compose, applyMiddleware } from 'redux';
import React from 'react';
import ReactDOM from 'react-dom';
import { AppContainer } from 'react-hot-loader';
import { JUGADORX, JUGADOR0, TABLERO } from '../constants/constants';
import App from './App';

export default class ReduxProvider extends React.Component {
  constructor(props) {
    super(props);
    this.initialState = {tablero: TABLERO, turno: JUGADORX, movimientos: 0};
    this.store = this.configureStore();
  }
  render() {
    return (
      <AppContainer>
       <Provider store={ this.store }>
        <div style={{ height: '100%' }}>
        <App store={ this.store } />
        </div>
        </Provider>
      </AppContainer>
    );
  }
configureStore() {
    const store = createStore(GlobalState, this.initialState);
    if (module.hot) {
        module.hot.accept('./../reducers/reducers', () => {
            const nextRootReducer =     require('./../reducers/reducers').default;
            store.replaceReducer(nextRootReducer);
        });
    }
    return store;
  }
}
```

main.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import ReduxProvider from './components/ReduxProvider';

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

render(ReduxProvider);
```

App.jsx

Añadimos dependencias

```
import { connect } from 'react-redux';
import { jugarPosicion } from './../reducers/actions';
import { JUGADORX, JUGADOR0, TABLERO } from '../constants/constants';
```

Cambiamos la declaración del componente

```
~~export default~~ class App extends React.Component {
```

Conectamos las props del componente <App/> con el estado de Redux

```
function mapStateToProps(state) {
    return {
        movimientos: state.movimientos,
        tablero: state.tablero,
        turno: state.turno
    };
}
export default connect(mapStateToProps)(App);
```

Ahora tenemos acceso a `this.props.movimientos, this.props.tablero` y `this.props.turno` en <App/>

Ejemplo de llamada a acción: `this.props.dispatch(jugarPosicion(0,0,JUGADOR_X))`

App.jsx

Borramos el estado inicial del constructor (ahora lo sacamos de Redux)

```
export default class App extends React.Component {
  constructor(props) {
    super(props);
    //this.state = {..}
    this.appClick = this.appClick.bind(this);
    this.resetClick = this.resetClick.bind(this);
  }
```

Cambiamos el método *render*

```
 render() {
      let texto = "Turno del " + this.state.turno this.props.turno;
      return (
        <div>
          <Cabecera texto={texto}/>
          <Tablero valores={this.state.valores this.props.tablero} appClick={this.appClick}/>
          <h3>Número de movimientos: {this.state.movimientos this.props.movimientos}</h3>
          <Reset resetClick={this.resetClick}></Reset>
        </div>
      );
  }
```

App.jsx

Modificamos el método *appClick(),* ahora toda la lógica pasa a los reducers

```
 appClick(numeroFila, numeroColumna) {
      let nuevosvalores = this.state.valores.slice();
      let nuevoValor = this.state.turno === JUGADORX ? 'X' : '0';
      nuevosvalores[numeroFila][numberoColumna] = nuevoValor;
      this.setState({
          turno: this.state.turno === JUGADORX ? JUGADOR0 : JUGADORX,
          valores: this.state.valores,
          movimientos: this.state.movimientos + 1
      });
      this.props.dispatch(jugarPosicion(numeroFila, numeroColumna, this.props.turno));
}
```

Cambiamos el método *resetClick(). Aún no tenemos acción de RESET!*

```
  resetClick(){
    this.setState({
      turno: JUGADORX,
      valores: TABLERO,
      movimientos: 0
    });
    console.log('RESET');
  }
```

App.jsx - completo

```
import React from 'react';
import './../assets/scss/main.scss';
import Cabecera from './Cabecera.jsx';
import Tablero from './Tablero.jsx';
import Reset from './Reset.jsx';
import { JUGADORX, JUGADOR0, TABLERO } from '../constants/constants';
import { connect } from 'react-redux';
import { jugarPosicion } from './../reducers/actions';

class App extends React.Component {
  constructor(props) {
    super(props);
    this.appClick = this.appClick.bind(this);
    this.resetClick = this.resetClick.bind(this);
  }
  appClick(numeroFila, numeroColumna) {
    this.props.dispatch(jugarPosicion(numeroFila, numeroColumna, this.props.turno));
  }
  resetClick(){
    console.log('RESET')
  }
 render() {
      let texto = "Turno del " + this.props.turno;
      return (
        <div>
          <Cabecera texto={texto}/>
          <Tablero valores={this.props.tablero} appClick={this.appClick}/>
          <h3>Número de movimientos: {this.props.movimientos}</h3>
          <Reset resetClick={this.resetClick}></Reset>
        </div>
      );
  }
}
function mapStateToProps(state) {
    return {
        movimientos: state.movimientos,
        tablero: state.tablero,
        turno: state.turno
    };
}
export default connect(mapStateToProps)(App);
```

Acción RESET

- Queremos que al pulsar el botón de **RESET** se reinicie el juego
- Para ello definimos una nueva acción y modificamos los reducers para que consideren el caso **RESET** y no sólo el **JUGAR_POSICION**
- No necesita parámetros

actions.jsx

```
export function jugarPosicion(x, y, turno) {
  return {
    type : 'JUGAR_POSICION',
    x : x,
    y : y,
    turno: turno
  }
}
export function reset(){
   return { type : 'RESET' }
}
```

gameReducer.jsx

Tenemos que vaciar el tablero

```
import {JUGADORX, JUGADOR0, TABLERO} from '../constants/constants';
function gameReducer(state = TABLERO, action) {
  switch (action.type) {
    case 'JUGAR_POSICION':
      let nuevoValor = action.turno === JUGADORX ? 'X' : '0';
      let newState = JSON.parse(JSON.stringify(state))
      newState[action.x][action.y] = nuevoValor;
      return newState;
   case 'RESET':
      return TABLERO;
    default:
      return state;
  }
}
export default gameReducer;
```

turnoReducer.jsx

Tenemos que cederle el turno al jugador inicial

```
import {JUGADORX, JUGADOR0} from '../constants/constants';

function turnoReducer(state = JUGADORX, action) {
  switch (action.type) {
    case 'JUGAR_POSICION':
      return action.turno === JUGADORX ? JUGADOR0 : JUGADORX;
    case 'RESET':
      return JUGADORX;
    default:
      return state;
  }
}
export default turnoReducer;
```

movimientosReducer.jsx

Resetear los movimientos a cero

```
function movimientosReducer(state = 0, action) {
  switch (action.type) {
    case 'JUGAR_POSICION':
      return state + 1;
    case 'RESET':
      return 0;
    default:
      return state;
  }
}
export default movimientosReducer;
```

App.jsx

Ahora importamos la nueva acción

```
import { jugarPosicion, reset } from './../reducers/actions';
```

Modificamos resetClick() para que lance la acción

```
resetClick() {
   this.props.dispatch(reset());
}
```

App.jsx - completo

```
import React from 'react';
import './../assets/scss/main.scss';
import Cabecera from './Cabecera.jsx';
import Tablero from './Tablero.jsx';
import Reset from './Reset.jsx';
import { JUGADORX, JUGADOR0, TABLERO } from '../constants/constants';
import { connect } from 'react-redux';
import { jugarPosicion } from './../reducers/actions';

class App extends React.Component {
  constructor(props) {
    super(props);
    this.appClick = this.appClick.bind(this);
    this.resetClick = this.resetClick.bind(this);
  }
  appClick(numeroFila, numeroColumna) {
    this.props.dispatch(jugarPosicion(numeroFila, numeroColumna, this.props.turno));
  }
  resetClick(){
    this.props.dispatch(reset);
 }
 render() {
      let texto = "Turno del " + this.props.turno;
      return (
        <div>
          <Cabecera texto={texto}/>
          <Tablero valores={this.props.tablero} appClick={this.appClick}/>
          <h3>Número de movimientos: {this.props.movimientos}</h3>
          <Reset resetClick={this.resetClick}></Reset>
        </div>
      );
  }
}
function mapStateToProps(state) {
    return {
        movimientos: state.movimientos,
        tablero: state.tablero,
        turno: state.turno
    };
}
export default connect(mapStateToProps)(App);
```

Cada vez que añadimos una acción

1. Definimos la acción en *actions.jsx*: *type* y parámetros
2. Modificamos los reducers para que tengan en cuenta la nueva acción
3. Lanzamos la acción desde App.jsx con *dispatch*

FIN