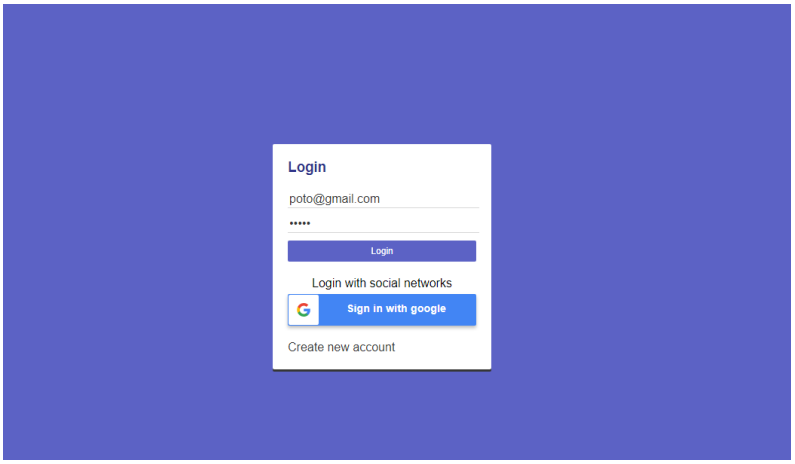
Resumen

Se utilizó como base la aplicación Journal App escrita en React de un curso de udemmy, para estudiar el modelo de login con el uso de redux



Objetivo

Documentar los conceptos y detalles relevantes aplicados en la arquitectura del sistema de logging ,con el uso de redux.

Antecedentes

Se necesita los conocimientos previos tratados en el ANEXO A - Conocimientos previos
El código fuente tratado en este documento se puede encontrar en el ANEXO A- Repositorio del código:

Introducción

Redux

Es un contenedor predecible del estado de nuestra aplicación , no es propio de react. Se puede usar en vue, angular o next.

Store

Aquí se encuentra la información que vamos a consumir.

El concepto de redux es similar al del reducer con el agregado de acciones asíncronas, donde se le agrega una capa middleware para el manejo de las mismas

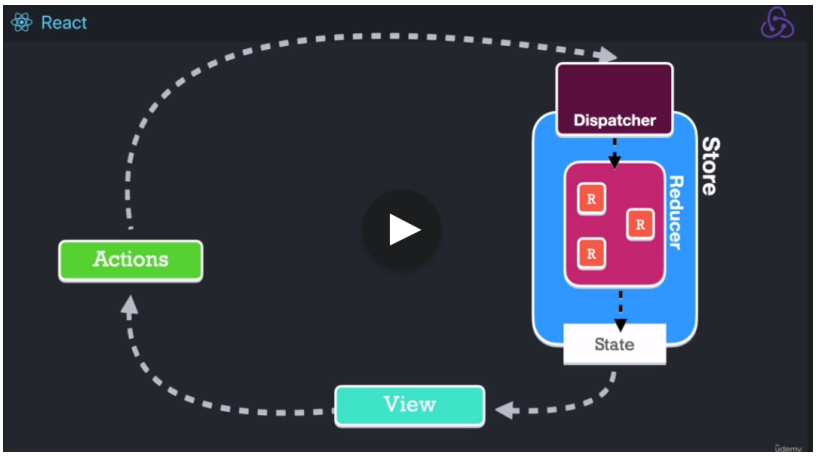


Figura 1: Diagrama de reducer, acciones síncronas.

Cuando la vista dispara una acción, el dispacher se lo envía al reducer, éste la ejecuta, actualiza el estado, y la vista al ser reactiva a este estado, se actualiza.
El dispatcher se lo envía a todos los reducers, pero como las acciones son únicas, solo el reducer que tenga la acción se ejecutara.

Cuando la acción que se dispara requiere de una petición asíncrona ,como puede ser un pedido de autorización del loggin, se agrega un middleware al modelo.
Este se encarga de resolver la tarea asíncrona con la api correspondiente y luego enviar la acción al reducer

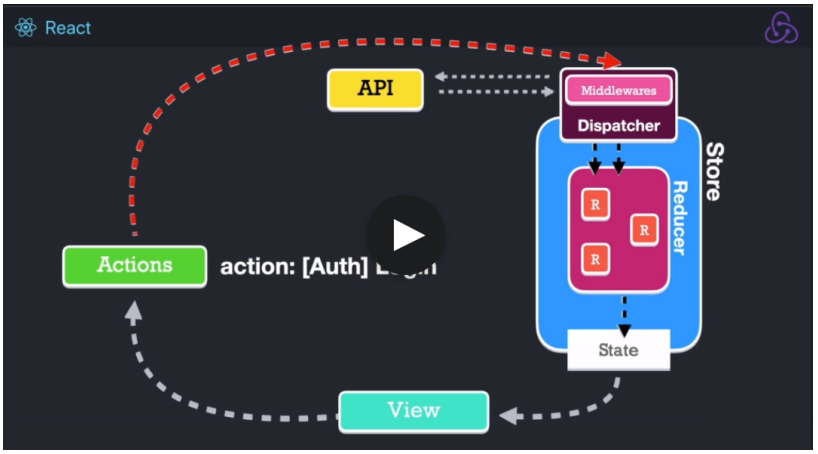


Figura 2: Diagrama de Redux, acciones asíncronas.

La configuración del store se implementa de la siguiente manera
store.js
import {createStore, combineReducers, applyMiddleware, compose } from 'redux'
import thunk from 'redux-thunk' //MiddleWare

```
import {createStore, combineReducers, applyMiddleware, compose } from
'redux'
import thunk from 'redux-thunk' //MiddleWare

// Custom reducer

import { authReducers } from '../reducers/authReduces';
import { uiReducer } from '../reducers/uiReduder';


// Configuracion

const composeEnhancers = (typeof window !== 'undefined' &&
window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__) || compose;


const reducers =combineReducers({
    auth: authReducers,
    ui:   uiReducer
});

export const store = createStore (
    reducers, // Solo recibe un deducer , por eso el uso de
combineReducers
    composeEnhancers(
        applyMiddleware(thunk)
    )
);
```

Para tener disponible el store, en toda la app, se importa en el punto más alto de la misma. JournaApp.js con el uso de Provider

```
import React from 'react';
import {Provider} from 'react-redux';

import { store } from './store/store';
import { AppRouter } from './routers/AppRouter';



export const JournalApp = () => {
    return (
        <Provider store ={ store}>
            <AppRouter />
        </Provider>
    )
}
```

Acciones

Para poder disparar la acción desde la vista se debe importar de redux, useDispacth y disparar la acción , esta última se crea en un archivo (actions/auth) diferente y se importa.

RegisterScreen.js

```
import { useDispatch, useSelector } from 'react-redux'
import { startRegisterWithEmailPasswordName } from '../../actions/auth'
export const RegisterScreen = () => {

    const dispatch = useDispatch();
const handleRegister = (e) => {
        e.preventDefault();

        if ( isFormValid () ){
            dispatch ( startRegisterWithEmailPasswordName( email,
password,name));
        }
    }
```

action/auth.js

```
export const startRegisterWithEmailPasswordName = ( email, password,
name) => {
    // Accion asincronica
firebase.auth().createUserWithEmailAndPassword
    return ( dispatch )=> { //el dispacht es provisto por el middleware
thunk cuando la tarea termina
        firebase.auth().createUserWithEmailAndPassword( email,
password)
        .then ( async ({user}) => {


            await user.updateProfile ( { displayName: name }) // Esto
es un promesa, se puede poner en otro .then
                                                                // se opto
por usar async

            //Accion sincronica login
            dispatch(
                login (user.uid, user.displayName)
            )

        })
        .catch ( e => { // Por si el usuario existe.
            console.log(e);
            Swal.fire('Error', e.message, 'error');
        })

    }
}
```

cabe destacar que la acción
startRegisterWithEmailPasswordName , dispara otra acción login.

dispatch(
login (user.uid, user.displayName)
)

Login es síncrona, definida también en actions/auth.js

```
export const login = (uid, displayName)=> {
    return {
        type: types.login,
        payload: {
            uid,
            displayName
        }
    }
}
```

para leer los estados del store desde cualquier lugar de la app

```
import { useDispatch, useSelector } from 'react-redux'
 const { name } = useSelector(state => state.auth);
```

nótese que state.auth es el nombre de un de nuestros reducer

```
const reducers =combineReducers({
    auth: authReducers,
    ui:   uiReducer
});
```

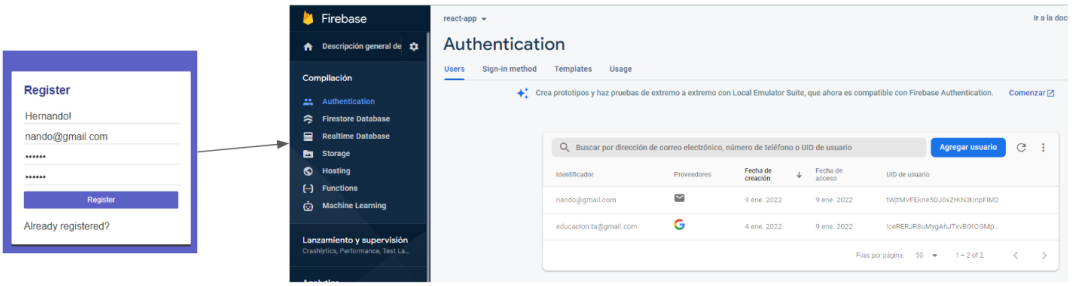
Conclusión: podemos disparar acciones asincronas y sincronas desde cualquier punto de la app, importando use dispatch, y agregando la nueva acción .

Modelo de login

RegisterScreen.js

El registro de usuario se hace en la base de datos firebase

Register



LoginScreen.js

El login puede hacer contra la base firebase o con la cuenta de email de google

Loggin

Reducers

Estado de usuario

Los estados del usuario se guardan en nuestro reducer auth

reducer/authReducer.js

```
import { types } from "../types/types";

/*
        Si se pudo autenticar
        {
            uid: 'asdasdsadsd'
            name: 'Fernando'
        }
        Si no esta autenticado
        {}
*/

export const authReducers = ( state ={}, action) =>{ // Siempre hay que
inicializar el state

  switch (action.type) {
        case types.login:
          return {
              udi:action.payload.uid,
              name:action.payload.displayName
          }

        case types.logout:
        return {}


      default:
          return state;
  }


}
```

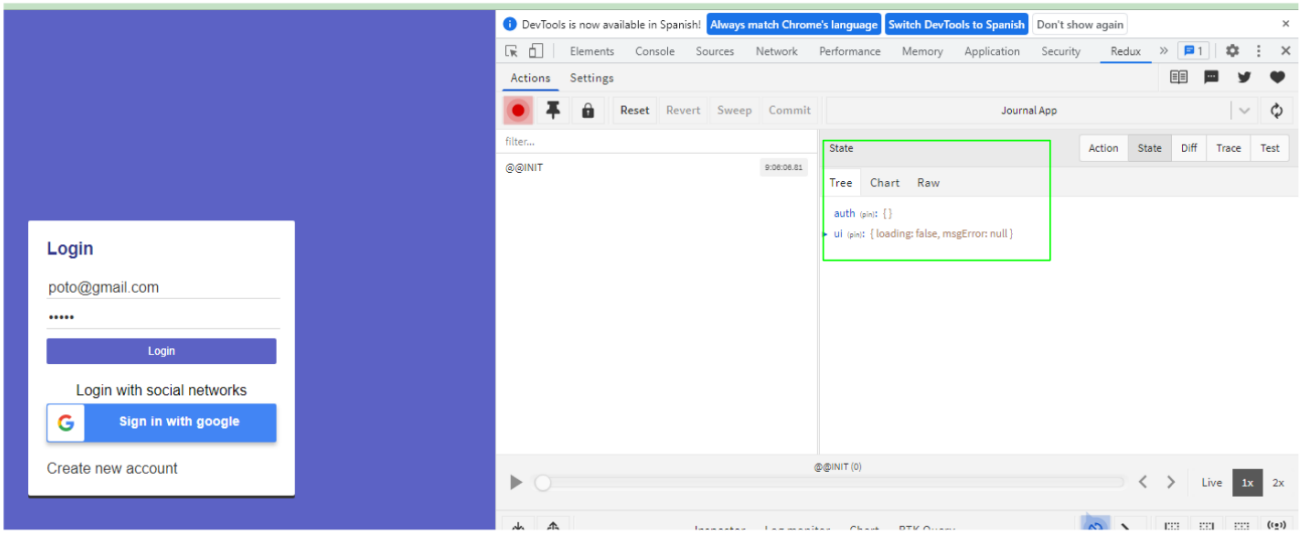
Si el usuario no está autenticado el estado es {} y si no devuelve el campo y el displayName

Estado de loggin

El loggin tiene un reducer ui.js para guardar un estado de cargando, y otro de error

```
import { types } from "../types/types";
export const setError = (err) => ({
    type: types.uidSetError,
    payload: err
});
export const removeError = () => ({
     type: types.uidRemoveError,
 });
 export const startLoading = () => ({
    type: types.uiStartLoading
});
export const finishLoading = () => ({
    type: types.uiFinishLoading
});
```

Estos estados se puede ver desde la dev tools redux



Desarrollo

Estados de log , error y acciones

Register

Estado de log

para manejar el estado de error de los campos del formulario se usa el estado del reducer

uiReducer : err
Se trae desde el store con el use de useSelector

```
const { msgError } = useSelector(state => state.ui);
```

y dependiendo de la validación se setea el mensaje y se dispara la acción deseada

```
const isFormValid = () => {

        if ( name.trim().length === 0){
            dispatch(setError('Nombre es requerido'));
            return false;
        }else if ( !validator.isEmail ( email)) {
            dispatch(setError('El email no es valido'));
            return false;
        }else if ( password !== password2 || password.length<6){
            dispatch(setError('password incorrecto, min 6'));

            return false;

        }
        dispatch(removeError());

        return true;
    }
```

Acciones

Las acciones están en uiReducer.js

```
import { types } from "../types/types";

const initialState = {
    loading: false,
    msgError: null
}
export const uiReducer = ( state = initialState, action ) => {
    switch (action.type) {
        case types.uidSetError:
            return {
                ...state,
                msgError: action.payload
            }
        case types.uidRemoveError:
        return {
            ...state,
            msgError: null
        }
        case types.uiStartLoading:
        return {
            ...state,
            loading: true
        }
        case types.uiFinishLoading:
        return {
            ...state,
            loading: false
        }
        default:
            return state;
    }
}
```

Si el formulario de registro es válido se dispara la función para iniciar el registro

```
 const handleRegister = (e) => {
        e.preventDefault();

        if ( isFormValid () ){
            dispatch ( startRegisterWithEmailPasswordName( email,
password,name));
        }
     }
```

Esta función dispara la acción asincrónica que dentro dispara la acción sincrónica loggin

```
export const startRegisterWithEmailPasswordName = ( email, password, name) => {
    // Accion asincronica firebase.auth().createUserWithEmailAndPassword
    return ( dispatch )=> { //el dispatch es provisto por el middleware thunk cuando la
tarea termina
        firebase.auth().createUserWithEmailAndPassword( email, password)
        .then ( async ({user}) => {
            await user.updateProfile ( { displayName: name }) // Esto es un promesa, se
puede poner en otro .then Se optó por usar async
            //Accion sincronica login
            dispatch(
                login (user.uid, user.displayName)
            )
        })
        .catch ( e => { // Por si el usuario existe.
            console.log(e);
            Swal.fire('Error', e.message, 'error');
        })
    }
}
```

Las acciones de log están definidas en action/auth.js

```
export const login = (uid, displayName)=> {

    return {
        type: types.login,
        payload: {
            uid,
            displayName
        }
    }
}

export  const logout = () => ({

    type: types.logout
})
```

y en types. los estados de los reducers authReducer.js

```
export const types ={

    login: '[Auth] Login',
    logout: '[Auth] Logout',

    uidSetError:  '[UI] Set error',
    uidRemoveError: '[UI] Remove error',

    uiStartLoading: '[UI] Start loading',
    uiFinishLoading: '[UI] Finish loading',
}
```

Login con la cuenta de email de google.

Acciones

La función que va a disparar las accion es

```
    const handleGoogleLogin=() =>{
        dispatch( startGoogleLogin ());
    }
```

```
export const startGoogleLogin = () => {
     // Accion asincronica  firebase.auth().signInWithPopup
    return ( dispatch ) => {
        firebase.auth().signInWithPopup( googleAuthProvider)
        .then ( ({user}) => {
            dispatch(
                //Accion sincronica login
                login (user.uid, user.displayName)
            )
        });
    }
}
```

Login con email y password con Firebase

Acciones

La función que va a disparar las accion es

```
const handleLogin = (e) => {
        e.preventDefault();
        dispatch(startLoginEmailPassword(email, password));
    }
```

```
export const startLoginEmailPassword = (email, password) =>{
    // Accion asincronica firebase.auth().signInWithEmailAndPassword(
email, password)
return(dispatch) => { // El dispatch lo ofrece thunk, le manda la
acciones a todos los reducers, como estas son unicas
                    // solo lo recibe el reducer correcto.

        dispatch ( startLoading ()); // Inicia el loading

        firebase.auth().signInWithEmailAndPassword( email, password)
        .then (  ({user}) => {
             //Accion sincronica login
            dispatch( login (user.uid, user.displayName))

            dispatch ( finishLoading ()); //Cancela el loading

        })
        .catch ( e => { // Por si el usuario no existe o el pass es
incorrecto.
            console.log(e);
            dispatch ( finishLoading () ); //Cancela el loading
            Swal.fire('Error', e.message, 'error');
        })
    }
```

Estado de log

Para `startLoginEmailPassword` se usa el estado de loading para indicar si el login termina o no.

Para esos se dispara una acción síncrona que setea el estado en true antes de comenzar el login y otra acción al terminar

```
 dispatch ( startLoading ()); // Inicia el loading

        firebase.auth().signInWithEmailAndPassword( email, password)
        .then (  ({user}) => {
             //Accion sincronica login
            dispatch( login (user.uid, user.displayName))

            dispatch ( finishLoading ()); //Cancela el loading

        })
```

Logout

```
export  const logout = () => ({
    type: types.logout
})
/*
* Asincrona
 si es Ok ejecuta el .then  si  no tira error
*/
export const startLogout = () => {

    return async ( dispatch )=> { // async Espero a que se ejecute
        firebase.auth().signOut();
        dispatch( logout() );
    }
}
```

Protección de rutas

Para evitar guardar en el local-session Storage o en un Cookies información “sensible” del usuario, uid y name se guardan solo en memoria , para poder mantener el estado al recargar el navegador y llamar la pantalla principal cuando el login es exitoso.

El appRouter permite el acceso a rutas publicas y privados dependiendo de un hook

```
const [isLoggedIn, setIsLoggedIn ] = useState( false ); // El usuario esta logeado?
```

```
  return (
        <Router>
            <div>
                <Switch>
                    <PublicRoute
                        isAuthenticated= { isLoggedIn }
                        path="/auth"
                        component={ AuthRouter }
                    />

                    <PrivateRoute
                        exact
                        isAuthenticated={ isLoggedIn }
                        path="/"
                        component={ JournalScreen }
                    />

                    <Redirect to="/auth/login" />


                </Switch>
            </div>
        </Router>
    )
```

El appRouter usa una función de firebase que genera un objeto observable al usuario de la base de datos, si este cambie dispara la función

```
 firebase.auth().onAuthStateChanged( (user)=> {

           /*
           * Si user existe "user?" entonces evalua user.id
           */
            if ( user?.uid ) {

                dispatch (login( user.uid, user.displayName));
                //console.log('observando user');
                setIsLoggedIn( true );

            }else {
                setIsLoggedIn( false );
            }

            setchecking( false ); // Termino de checkear firebase.
        });
```

que determine el estado del hook `isLoggedIn`

Con esta estrategia cada vez que se recarga el navegador o se cambie el usuario autenticado en firebase, la app sabe donde debe routear el acceso al usuario.

appRouter.js

```
import { useDispatch } from 'react-redux';

import { firebase } from '../firebase/firebase-config';
import { AuthRouter } from './AuthRouter';
import { PrivateRoute } from './PrivateRoute'
import { PublicRoute } from './PublicRoute'

import { JournalScreen } from '../components/journal/JournalScreen';
import { login } from '../actions/auth';


export const AppRouter = () => {
```

```
  const dispatch = useDispatch();
  const [ checking, setchecking ] = useState( true );      // Esta
chequeando el estado de firbase
  const [isLoggedIn, setIsLoggedIn ] = useState( false ); // El usuario
esta logeado?

// Esta observando si la auth cambia y la actualiza
    useEffect(() => {

        // Esto crea un objeto observable lo llama user, pero es igual
a la autorizacion de firebase, si esta cambia se dispara esto

        firebase.auth().onAuthStateChanged( (user)=> {

           /*
           * Si user existe "user?" entonces evalua user.id
           */
            if ( user?.uid ) {

                dispatch (login( user.uid, user.displayName));
                //console.log('observando user');
                setIsLoggedIn( true );

            }else {
                setIsLoggedIn( false );
            }

            setchecking( false ); // Termino de checkear firebase.
        });
    // Como el useDispacth puede cambiar el useEffect tira un warning
,para evita los ponemos como dependencia

    }, [ dispatch, setchecking, setIsLoggedIn ]) // Todo lo que cambie
dentro del useEffect es una dependencia.


    /*
    * Se queda aca hasta que cambie checking
    */
    if ( checking ){
        return (
            <h1> Espere...</h1>
```

```
        );
    }


    return (
        <Router>
            <div>
                <Switch>
                    <PublicRoute
                        isAuthenticated= { isLoggedIn }
                        path="/auth"
                        component={ AuthRouter }
                    />

                    <PrivateRoute
                        exact
                        isAuthenticated={ isLoggedIn }
                        path="/"
                        component={ JournalScreen }
                    />

                    <Redirect to="/auth/login" />


                </Switch>
            </div>
        </Router>
    )
}
```

Anexo A

Conocimiento previos: https://github.com/theinsideshine/react-todoApp

Repositorio del código: https://github.com/theinsideshine/react-reduxApp