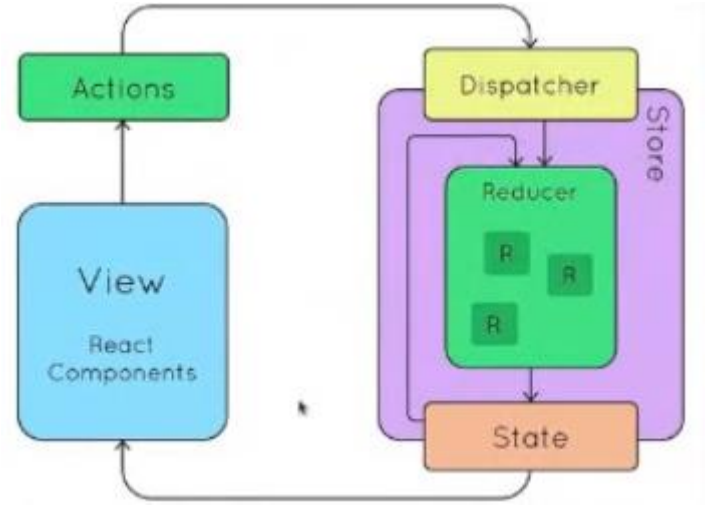
Como funciona redux???

Redux es una librería JavaScript que emite actualizaciones de estado en respuesta a acciones, con la peculiaridad de realizar dichas modificaciones a través de objetos sencillos, que reciben el nombre de acciones, y no a través de cambios directos en el estado.

REDUX THUNK: es un middleware de redux que te permiten hacer que tus action creators sean asíncronas, porque las consultas a las APIS deben ser asincrónicas y hace que pueda usar promises, async await.

El Action pasa al Store que es donde se guardan todos los estados. El Store comunica al Reducer el estado actual y cual fue el Action que se ejecutó. Luego, el Reducer retorna un nuevo estado modificado por la acción que se acaba de ejecutar. El estado se actualiza en el Store .



Ciclos de vida!!!!

https://es.reactjs.org/docs/state-and-lifecycle.html

Que cambien async por fetch

Async

```
export function searchGames(){
    return async function(dispatch){
        let games = await axios.get('/videogames')
        return dispatch({
            type: buscar,
            payload: games.data
        })
    }
}
```

Fetch

```
export function searchGames(){
```

```
    return function(dispatch){
        return fetch('http://localhost:3001/api/videogames')
        .then(r => r.json())
        .then(json => {
            return dispatch ({
                type: buscar,
                payload: json
            })
        })
    }
}
```

Función de dispatch:

https://medium.com/react-redux/glosario-de-t%C3%A9rminos-de-redux-c2bca005ca69

type BaseDispatch = (a: Action) => Action

type Dispatch = (a: Action | AsyncAction) => any

La función despachadora (o simplemente función dispatch) es una función que acepta una acción o una acción asíncrona; entonces puede o no despachar una o más acciones al store.

Debemos distinguir entre una función despachadora en general y la función base dispatch provista por la instancia del store sin ningún middleware.

La función base dispatch siempre envía síncronamente acciones al reducer del store, junto al estado anterior devuelto por el store, para calcular el nuevo estado. Espera que las acciones sean objetos planos listos para ser consumidos por el reducer.

Los middlewares envuelven la función dispatch base. Le permiten a la función dispatch manejar acciones asíncronas además de las acciones. Un middleware puede transformar, retrasar, ignorar o interpretar de cualquier forma una acción o acción asíncrona antes de pasarla al siguiente middleware. Lea más abajo para más información.

Cambiar la ruta de id cambiar parameters por query y en el Front le agregar ?=

```
En el Back por params
router.get('/:id', async (req, res) => {
    const { id } = req.params;
    try {
        if(!Number(id)){
            let juegoId = await Videogame.findOne({
                where: {
                    id
```

```
                },
                include:{
                    model: Genre,
                    attributes: ['name'],
                    through:{
                        attributes: []
                    }
                }
            })
            return res.json(juegoId)
        }
        let gameId = await getApiById(id)
        return res.json(gameId);
    }
    catch(e){
        res.send('Id no encontrado')
    }
})

En el Back por query→ usar la route de todos los juegos!!!
router.get('/', async (req, res) => {
    const { id } = req.query;
    if(id){
        if(!Number(id)){
            let juegoId = await Videogame.findOne({
                where: {
                    id
                },
                include:{
                    model: Genre,
                    attributes: ['name'],
                    through:{
                        attributes: []
                    }
                }
            })
            return res.json(juegoId)
        }
        let gameId = await getApiById(id)
        return res.json(gameId);
    }
    const { name } = req.query
    try {
        if (name) {
            const infoByName = await getInfoByName(name)
            res.json(infoByName);
        }
        else{
            const allDate = await getAllInfo()
```

```
            res.json(allDate);
        }
        }
        catch(e){
            res.send('Juego no encontrado')
        }
})


En el Frond por params
export function gameById(id){
    return async function(dispatch){
        let gamesId = await axios.get(`/videogames/${id}`)
        return dispatch({
            type: buscarNombrePorId,
            payload: gamesId.data
        })
    }
}

En el Frond por query
export function gameById(id){
    return async function(dispatch){
        let gamesId = await axios.get(`/videogames?id=${id}`)
        return dispatch({
            type: buscarNombrePorId,
            payload: gamesId.data
        })
    }
}
```

Si pide hacer un listado de Plataforma:

API

Models

sequelize.define('platforms', {

  name: {

   type: DataTypes.STRING,

   allowNull: false,

   primaryKey: true

  }

```
 })
Router
const { Router } = require('express');
const axios = require('axios');
const { key } = require('../../config-env/config');
const { Videogame, Platforms } = require('../../db.js');
const { v4: uuidv4 } = require('uuid');

const router = Router();




//Plataforms:
router.get('/', async (req, res) => {

   try {

      //Esto comentado es para crear las plataformas dentro de la base de datos:
      // let platforms = axios.get(`https://api.rawg.io/api/platforms/lists/parents?key=${key}`)
      // // platforms = platforms.data.results

      // let platformsPage2 = axios.get(`https://api.rawg.io/api/platforms?key=${key}`)
      // // platformsPage2 = platformsPage2.data.results

      // let platformsPage3 = axios.get(`https://api.rawg.io/api/platforms?key=${key}&page=2`)
      // // platformsPage3 = platformsPage3.data.results

      // const datos = await Promise.all([platforms, platformsPage2, platformsPage3])

      // platforms = datos[0].data.results;
      // platformsPage2 = datos[1].data.results;
```

```
        // platformsPage3 = datos[2].data.results;

        // platforms = platforms.concat(platformsPage2).concat(platformsPage3)

        // platforms = platforms.map(e => {
        //     return {
        //         id: e.id,
        //         name: e.name
        //     }
        // })

        // platforms.forEach(e => {
        //     Platforms.findOrCreate({
        //         where:{
        //             name: e.name
        //         }
        //     })
        // })

        const plataformas = await Platforms.findAll()

        res.json(plataformas);
    }
    catch(e){
        res.status(404).json({message: "No hay plataformas dentro de la base de datos"});
    }
})

module.exports = router
```

conexiones

```
Videogame.belongsToMany(Platforms, {through: 'videogamesPlatforms'});
```

Platforms.belongsToMany(Videogame, {through: 'videogamesPlatforms'});

POST

createdGame.addPlatforms(arrPlatforms.length > 0 ? arrPlatforms : platforms)

CLIENT

// export function getPlatforms(){

// return async function(dispatch){

// let platforms = await axios.get('http://localhost:3001/api/platforms')

// return dispatch({

// type: obtenerPlataformas,

// payload: platforms.data

// })

// }

// }

Create

<label>Plataformas</label>

{

platforms && platforms.map(e => <input type="checkbox" name="platforms" onChange={(evt)=>handlePlatforms(evt)} value={e.name} /><label>{e.name}</label>)

}

---

const dispatch = useDispatch();

const allVideoGames = useSelector((state) => state.videoGames)

reemplaza

```
const mapStateToProps = (store) => {
    return{
        videogames: store.videoGames,
```

```
        name: store.name,
    }
}

export default connect(mapStateToProps, {searchGames, gameById})(Card);
```

useSelector----→ mapStateToProps

dispach--→ {searchGames, gameById}

Rutas otra forma de armarla

```
<Route exact path= '/app/home'>
            <Nav />
            <Order />
            <Card />
            <Paginado />
          </Route>

          <Route exact path='/app/home/create'>
            <Nav />
            <Create />
          </Route>

          <Route exact path='/'>
            <LandingPage />
          </Route>

          <Route exact path='/app/:id'>
            <Nav />
            <Detail />
          </Route>
```

DELETE (back)

Crear ruta en Index

router.delete('/delete', deleteActivity);

videogame / Activity

const deleteActivity = async (req, res) => {

  try {

```
    const { name } = req.query;
     //validaciones

    await Activity.destroy({
      where: {
        name: name
      }
    });
    return res.status(200).send("Activity deleted");
  } catch (e) {
    console.error(e);
    return res.status(404).json({ message: "Creation not deleted" });
  }
};
```

(Frond)

Action

```
export const deleteActivity = (payload) => {
  return async function (dispatch) {

    try {
      const response = await axios.delete('http://localhost:3001/activity/' + payload);
      return dispatch({
        type: 'REMOVE_ACTION',
        payload: response.data
      });
    } catch (e) {
      console.error(e);
    }
  };
};
```

Reducer

```
case 'REMOVE_ACTION':
```

```
  const deleteActivity = state.activities;
 const deletea = deleteActivity.filter(a=> a !== action.payload)
 return {
  ...state,
  activities: deletea
 }
  default:
   return state;
  };
```

delete en donde quieran (videogame o genres)

```
{countryFound.Activities?.map((a) => {
            return (<div>
              <h3>Name: {a.name}</h3>
              <h4>Difficulty: {a.difficulty}</h4>
              <h4>Duration: {a.duration} min.</h4>
              <h4>Season: {a.season}</h4>
              <button onClick= {()=> deleteAction(a.name)}>DELETE</button>
            </div>
          )})}
```

la logica del dispatch

```
function deleteAction(name){
  dispatch(deleteActivity(name))
 }
```

PUT(generos) Modifica

Crear Ruta en Index

```
router.use('/api/modificar', routerModificar)
```

videogame

```
router.put('/', async (req, res)=>{
try{
```

```
const {name, newName} =req.body;
await Genre.update({ name: newName },
 { where: {
 name: name
}
 });
return res.status(200).send("Genre is updated")
} catch(e){
 return res.status(404).json({message:"Genre isn't updated"})
 }
})
```

PUT(date) Modifica

Crear Ruta en Index

```
router.use('/api/modificar', routerModificar)
```

videogame

```
const { Router } = require('express');
const axios = require('axios');
const { Videogame } = require('../../db.js');
const router = Router();
router.put('/:id', async (req, res) => {
   const { id } = req.params;
   const { name, description } = req.body;
   if(!Number(id)){
      let game = await Videogame.findOne({
         where: {
            id: id
         }
      })
      game = await game.update({
```

```
            name: name,
            description: description
        })
        res.json(game)
    }
    res.status(404).json({message: "id no encontrado"})
})
module.exports = router
```