

REDUX

STORE

```
JS index.js ...\store X JS index.js ...\actions JS index.js ...\routes
client > src > redux > store > JS index.js > default
1 import { applyMiddleware, createStore } from 'redux';
2 import { composeWithDevTools } from 'redux-devtools-extension';
3 import thunk from 'redux-thunk';
4 import rootReducer from '../reducer';
5
6 const store = createStore(
7   rootReducer,
8   composeWithDevTools(applyMiddleware(thunk))
9 );
10
11 export default store;
```

Importamos las herramientas que necesitamos de las dependencias instaladas, como las devtools de redux que nos permiten tener más visibilidad de los procesos de redux que se van ejecutando a medida que realizamos algún tipo de acción, a su vez importamos un createStore, para crear el que será el almacenamiento de la “verdad absoluta” de los estados globales de redux y recibirá como parámetro todo los reducer, y el middleware a usar que será thunk, de redux-thunk. (explicacion)

INDEX DE CLIENT

```
EXPLORER ... JS index.js X
client > src > JS index.js
1 import React from 'react';
2 import ReactDOM from 'react-dom';
3 import './index.css';
4 import App from './App';
5 import reportWebVitals from './reportWebVitals';
6 import { Provider } from 'react-redux';
7 import store from './redux/store';
8
9
10 ReactDOM.render(
11   <Provider store={store}>
12     <React.StrictMode>
13       <App />
14     </React.StrictMode>
15   </Provider>,
16   document.getElementById('root')
17 );
```

envolvemos toda la aplicación (componente App) con el provider que se encargará de entregarnos los valores asignados en los estados globales de redux, dentro del store

ACTIONS

```

import axios from "axios";
import imageError404 from "../components/Home/images/not-found-404.png";
const urlApi = "http://localhost:3001";

export function getAllCountries() {
  return async function(dispatch) {
    let json = await axios.get(`${urlApi}/countries`);
    return dispatch({
      type: "GET_ALL_COUNTRIES",
      payload: json.data
    });
  };
}

export function getAllActivities() {
  return async function(dispatch) {
    let json = await axios.get(`${urlApi}/activity`);
    return dispatch({
      type: "GET_ALL_ACTIVITIES",
      payload: json.data,
    });
  };
}

```

La función `getAllCountries` por medio de una función asíncrona obtiene todos los países de la base de datos enviados por el api y envía esos datos como payload y también envía un tipo de acción llamado "GET_ALL_COUNTRIES". Y de la misma manera con la función `getAllActivities`.

```

export function getDogs() {
  return async function(dispatch){
    var json = await axios.get("http://localhost:3001/dogs",{
    });
    return dispatch({
      type: "GET_DOGS",
      payload: json.data
    })
  }
}

```

Eso es posible solamente gracias a que se está usando el middleware Thunk. Nativamente redux no permite ningún tipo de acción asíncrona ni en el store, ni el reducer, ni en las actions. El thunk permite despachar una asincronía que con su response ejecuta a su vez una dispatch normal.

Dispatch

Las acciones tienen que ser enviadas al Store para que surgan efecto. La función `dispatch` base es un método de store, esta manda una acción de forma sincrónica a los reducers del store, junto con el estado previo retornado por el store, para calcular el nuevo estado. Espera que las acciones que les pasemos sean objetos planos, listas para ser consumidas por los reducers. También se puede envolver a los dispatchers en una serie de Middlewares, esto permite que los dispatchers puedan manejar acciones asíncronas, además de poder transformar, demorar, ignorar o interpretar acciones antes de pasarla al siguiente Middleware.

```

export function getCountryByName(name) {
  return async function(dispatch) {
    try {
      let json = await axios.get(`${urlApi}/countries?name=${name}`);
      return dispatch({
        type: "GET_COUNTRY_BY_NAME",
        payload: json.data,
      })
    } catch (error) {
      // console.log(error);
      return dispatch({
        type: "GET_COUNTRY_BY_NAME",
        payload: [{id: "error404", name: "COUNTRY DON'T FOUND", img_flag: imageError404}],
      })
    }
  }
}

export function getByID(idCountry) {
  return async function(dispatch) {
    try {
      let json = await axios.get(`${urlApi}/countries/${idCountry}`);
      return dispatch({
        type: "GET_COUNTRY_BY_ID",
        payload: json.data,
      });
    } catch (error) {
      console.log(error);
    }
  }
}

```

En la función `getCountryByName()` recibe por parámetro el nombre del país a buscar y lo enviara por query, usamos un try catch para capturar errores, y enviara la data como payload, y enviara un type eso en caso exitoso, en caso de falla enviara como payload un arreglo con un id, name y una imagen de bandera que yo he seleccionado con mensaje de error 404 para informar al usuario.

En la función `getById()` recibirá por parámetro el id del país y lo enviara como params, la información obtenida de esa solicitud se enviara como payload, y se usará para mostrar la info de sus detalles.

```

57 export function resetDetailById() {
58   return{
59     type: "RESET_DETAIL_BY_ID",
60     payload: [],
61   }
62 }
63
64 export function orderByName(order) {
65   return{
66     type: "ORDER_BY_NAME",
67     payload: order
68   }
69 };
70
71 export function orderByPopulation(order) {
72   return{
73     type: "ORDER_BY_POPULATION",
74     payload: order,
75   }
76 }
77
78 export function filterByContinent(continentName) {
79   return{
80     type: "FILTER_BY_CONTINENT",
81     payload: continentName
82   }
83 }

```

La función `resetDetailById()` hace que al dar click en el home el valor del país en el estado actual sea vacío, porque sucedía que había como un bug al buscar los detalles de un país te mostraba el país anterior por un pequeño lapso de tiempo y luego cargaba el nuevo.

Función `orderByByName()` y `orderByPopulation()` recibe parámetro el orden de ordenamiento y lo pasa como payload junto con un type

Function `filterByContinent()` y `filterByActivity()` recibe un parametro a evaluar y si cumple dicha condición se agregará en el estado de países a mostrar, dicho parámetro lo retornará como su payload y también con un type.

```
export function filterByActivity(activity) {
  return {
    type: "FILTER_BY_ACTIVITY",
    payload: activity
  }
}

export const postActivity = (payload) => {
  return async function() {
    const res = await axios.post("http://localhost:3001/activity", payload);
    return res;
    // return response.data;
  };
};
```

Función `postActivity()` recibe un parámetro en este caso llamado `payload`, usamos una función asíncrona, para ejecutar el método `post` en la url especificada dentro del primer parámetro del `post` y como segundo parámetro que será enviado por body ira el `payload` de los valores a crear en la base de datos.

Ya que el usuario será almacenado en la base de datos no será necesario realizar acciones en el reducer porque ese país será tomado cuando se despache la acción `getAllCountries()`

REDUCERS

```
PI-COUNTRIES-MAIN
├── api
├── client
├── node_modules
├── public
├── src
│   ├── components
│   ├── redux
│   │   ├── actions
│   │   └── reducer
│   │       └── index.js
│   ├── store
│   │   └── index.js
│   ├── App.css
│   ├── App.js
│   ├── App.test.js
│   ├── index.css
│   ├── index.js
│   ├── reportWebVital...
│   ├── setupTests.js
│   ├── .env
│   ├── package-lock.json
│   ├── package.json
│   ├── .gitignore
│   ├── countries.png
│   └── README.md
├── OUTLINE
└── TIMELINE

client > src > redux > reducer > index.js > initialState

1  const initialState = {
2      totalData: [],
3      countries: [],
4      allCountries: [],
5      allActivities: [],
6      details: [],
7      continents: [],
8  }
9
10 const rootReducer = (state = initialState, action) => {
11     switch (action.type) {
12         case "GET_ALL_COUNTRIES":
13             return{
14                 ...state,
15                 countries: action.payload,
16                 allCountries: action.payload,
17                 totalData: action.payload,
18             }
19         case "GET_ALL_ACTIVITIES":
20             return{
21                 ...state,
22                 allActivities: action.payload,
23             }
24         case "GET_COUNTRY_BY_NAME":
25             return{
26                 ...state,
27                 countries: action.payload,
28             }
29         case "GET_COUNTRY_BY_ID":
30             return{
31                 ...state,
32                 details: action.payload,
33             }
34         case "RESET_DETAIL_BY_ID":
35             return{
36                 ...state,
37                 details: action.payload
38             }
39         case "ORDER_BY_NAME":
40             console.log("reducer");
```

Establecemos un estado inicial el cual se le pasara como parámetro igualándolo al estado de rootReducer, esto valores serán un estado de la data con los países la cual no se modificará y solo se obtendrá de la DB, se tendrá un valor para los países, las actividades, detalles, y países por continente.

...al hacer spring operator devolvemos el estado anterior. Un nuevo estado es generado, resultado de combinar el viejo estado y la acción del usuario. Este proceso lo realiza una función llamada reducer. Las describen que algo pasó, pero no especifican cómo cambió el estado de la aplicación en respuesta. Esto es trabajo de los reducers.

```

39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
public
src
  components
  redux
    actions
    reducer
      index.js
  store
    index.js
  App.css
  App.js
  App.test.js
  index.css
  index.js
  reportWebVital...
  setupTests.js
.env
package-lock.json
package.json
.gitignore
countries.png
README.md
OUTLINE
TIMELINE
case "ORDER_BY_NAME":
  console.log("reducer");
  console.log(action.payload);
  const sortedName = action.payload === "A-Z"
    ? state.allCountries.sort((a, b) => {
        if (a.name > b.name) {
          return 1;
        }
        if (b.name > a.name) {
          return -1;
        }
        return 0;
      })
    : state.allCountries.sort((a, b) => {
        if (a.name > b.name) {
          return -1;
        }
        if (b.name > a.name) {
          return 1;
        }
        return 0;
      })
  console.log(sortedName);
  return {
    ...state,
    countries: sortedName,
  };

```

En el caso “ORDER_BY_NAME” usamos un operado ternario para validar el tipo de orden dispatchado, para ordenar los elementos usamos el método sort donde se agregan dos parámetros comparadores que en este caso serán a y b, en caso de retornar un valor negativo quiere decir que el valor a va antes que el valor b y si devolvemos un positivo el valor b va antes que el valor a y si retorno el valor 0 los elementos se mantendrán en la misma posición.

```

client > src > redux > reducer > index.js > rootReducer
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
PI-COUNTRIES-MAIN
  api
  node_modules
  src
    models
    routes
      index.js
      app.js
      db.js
    tests
    .env
    .eslintrc.json
    .npmrc
    index.js
    package-lock.json
    package.json
  client
    node_modules
    public
    src
      components
      redux
        actions
        reducer
          index.js
      store
        case "ORDER_BY_POPULATION":
          const sortedPopulation = action.payload === "min-p"
            ? state.allCountries.sort((a, b) => {
                if (parseInt(a.population) > parseInt(b.population)) {
                  return 1;
                }
                if (parseInt(b.population) > parseInt(a.population)) {
                  return -1;
                }
                return 0;
              })
            : state.allCountries.sort((a, b) => {
                if (parseInt(a.population) > parseInt(b.population)) {
                  return -1;
                }
                if (parseInt(b.population) > parseInt(a.population)) {
                  return 1;
                }
                return 0;
              })
          return {
            ...state,
            countries: sortedPopulation,
          }
          //se puede hacer que al ejecutarse la accion llame el valor total de paises
        case "FILTER_BY_CONTINENT":

```

Hacemos el ordenamiento con el método sort como previamente se había explicado y el resultado se convertirá en el valor del estado de countries.

```
client > src > redux > reducer > .js index.js > rootReducer
91 //se puede hacer que al ejecutarse la accion llame el valor total de paises
92 case "FILTER_BY_CONTINENT":
93   let filterContinent;
94   if (action.payload === "All") {
95     filterContinent = state.totalData;
96     state.allCountries = state.totalData
97   } else{
98     filterContinent = state.totalData.filter(country => country.continent === action.payload);
99   }
100   return{
101     ...state,
102     countries: filterContinent,
103     allCountries: filterContinent
104   }
```

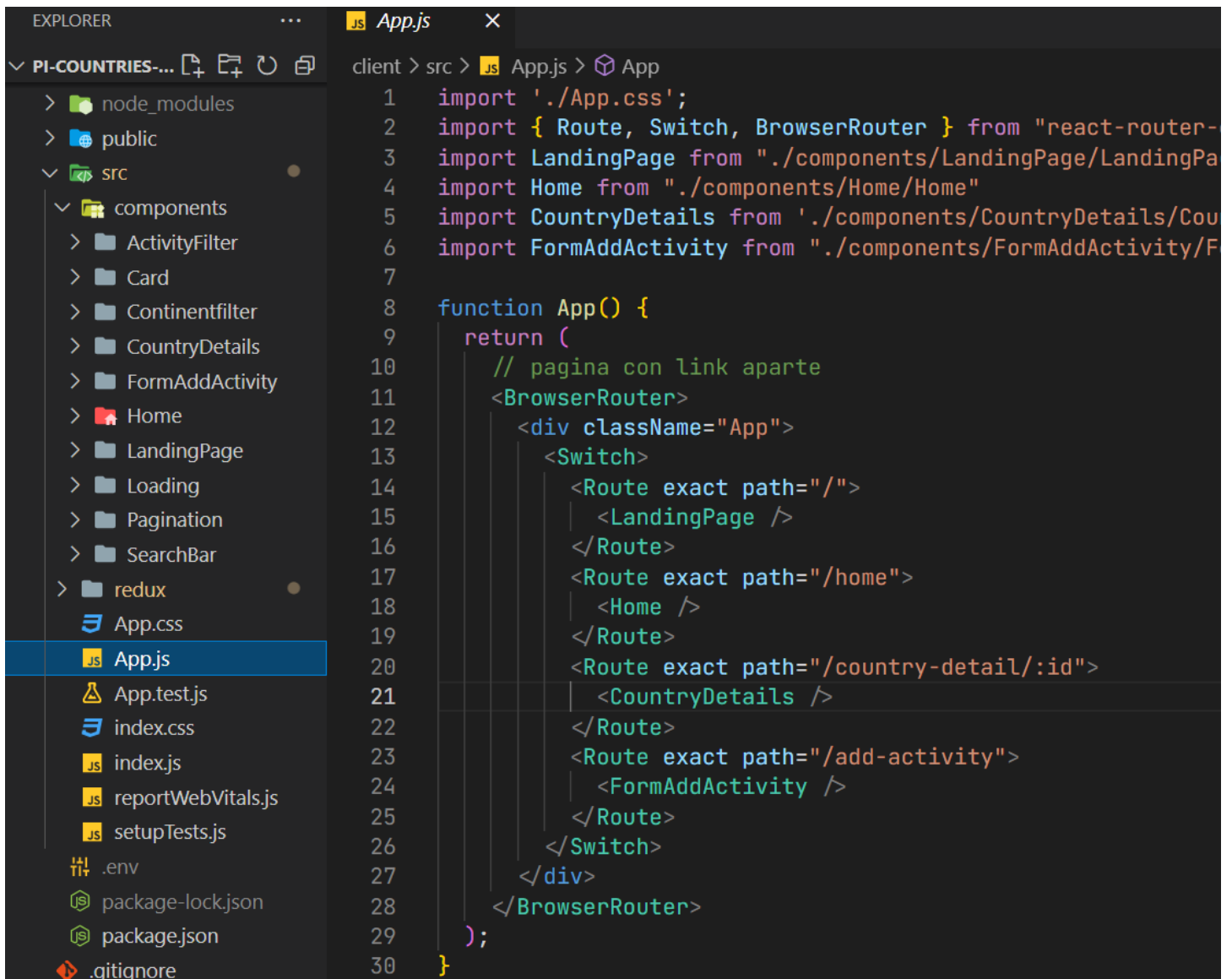
Filtrado por continente, si selecciono la opción de todos los continentes en países filtrados me pase la info de todos los países, en caso de elegir una opción distinta use el método filter() para filtrar todos los países que su continente coincida con con el enviado por el payload.

```
client > src > redux > reducer > .js index.js > rootReducer
104 }
105 case "FILTER_BY_ACTIVITY":
106   //reset para mostrar todas las actividdes
107   if (action.payload === "All") {
108     return{
109       ...state,
110       countries: state.totalData
111     }
112   }
113   const filteredCount = state.allCountries.filter(el => el.activities[0]);
114   console.log(action.payload);
115   let selectedCount = [];
116   filteredCount.forEach(elem => {
117     for (let i = 0; i < elem.activities.length; i++) {
118       if (elem.activities[i].id === parseInt(action.payload)) {
119         selectedCount.push(elem)
120       }
121     }
122   });
123   return{
124     ...state,
125     countries: selectedCount
126   }
127 default:
128   return state;
129 }
130 };
```

Si el payload es "All" el estado de countries serán todos los países. De lo contrario primero validamos y filtramos los países con una actividad asociada. Luego en el forEach iteramos el array con países con actividad asociada donde recorremos todo el subarray de las actividades que contiene y validamos por medio del id si alguna de ellas coincide con la solicitada, que fue enviada por medio del payload, si es así pusheamos el país en un arreglo que al final ese arreglo será el valor del estado de countries.

Por default en caso de que el caso no haya sido similar a alguno de los especificados retornamos el estado actual.

COMPONENTES



```
1  import './App.css';
2  import { Route, Switch, BrowserRouter } from "react-router-dom";
3  import LandingPage from "../components/LandingPage/LandingPage";
4  import Home from "../components/Home/Home";
5  import CountryDetails from "../components/CountryDetails/CountryDetails";
6  import FormAddActivity from "../components/FormAddActivity/FormAddActivity";
7
8  function App() {
9    return (
10      // pagina con link aparte
11      <BrowserRouter>
12        <div className="App">
13          <Switch>
14            <Route exact path="/">
15              <LandingPage />
16            </Route>
17            <Route exact path="/home">
18              <Home />
19            </Route>
20            <Route exact path="/country-detail/:id">
21              <CountryDetails />
22            </Route>
23            <Route exact path="/add-activity">
24              <FormAddActivity />
25            </Route>
26          </Switch>
27        </div>
28      </BrowserRouter>
29    );
30  }
```

BrowserRouter es un componente que nos entrega react-router-dom que nos permite establecer rutas en nuestra aplicación, según el componente que deseamos renderizar le especificamos en que ruta debe hacerlo, entonces dentro del componente principal de la aplicación lo envolvemos con BrowserRouter,

BrowserRouter (Router): inyecta propiedades a nuestro componente para poder acceder al historial de navegación, realizar redirecciones, etc.

Usamos Switch que nos permite especificar que cuando encuentre cierta ruta renderice dicho componente, y para especificar la ruta usamos Route y le damos el un camino especificado usando path=""


```

1  import React from "react";
2  import { Link } from "react-router-dom";
3  import S from "../LandingPage/LandingPage.module.css";
4
5  function LandingPage() {
6
7      document.title = "Welcome";
8      return(
9          <div className={S.main_container}>
10             <div className={S.elements_container}>
11                 <div className={S.text_container}>
12                     <h1>Country web application</h1>
13                     <p>Obtain information from different countries while you learn about their tourist activities</p>
14                 </div>
15                 <Link to="/home">
16                     <button className={S.button_home}>
17                         <span>Home</span>
18                     </button>
19                 </Link>
20             </div>
21         </div>
22     )
23 }
24

```

Componente landing page, un texto de bienvenida, un enlace que te envía a la ruta home y una imagen de fondo. Usamos css modules porque a mi parecer me parece más cómodo de manejar y es muy similar a darle estilos y agregar la clase del estilo al css y html convencional además que me permite usar pseudo clases como hover, focus y pseudo elementos como before y after.

```

client > src > components > Home > Home.jsx > ...
1  import React from "react";
2  import { useState, useEffect } from "react";
3  import { useDispatch, useSelector } from "react-redux";
4  import { Link } from "react-router-dom";
5  import S from "../Home/Home.module.css";
6  import SearchBar from "../SearchBar/SearchBar";
7  import Card from "../Card/Card";
8  import ContinentFilter from "../Continentfilter/Continentfilter";
9  import ActivityFilter from "../ActivityFilter/ActivityFilter";
10 import Pagination from "../Pagination/Pagination";
11 import Loading from "../Loading/Loading";
12
13 import {
14     getAllCountries,
15     getAllActivities,
16     orderByName,
17     orderByPopulation
18 } from "../../redux/actions"
19

```

En el componente Home importamos los hooks useState, UseEffect, useDispatch, useSelector, Link, los estilos de css a usar, también otros componentes a agregar y finalmente importamos las acciones a usar en el componente home.

```
function Home() {

  const dispatch = useDispatch();
  const allActivities = useSelector(state => state.allActivities);
  const countries = useSelector(state => state.countries)

  // console.log(countries);

  // eslint-disable-next-line no-unused-vars
  const [update, setUpdate] = useState("");
  const [currentPage, setCurrentPage] = useState(1);
  const countriesPerPage = 10;
  const lastIndex = currentPage * countriesPerPage; //2 * 10 = 20 ** 2 * 10 -1 = 19
  const firstIndex = lastIndex - countriesPerPage; //20 - 10= 10 ** 20 - 10 -1 = 9
  //si la pagina actual es 1 muestre los primeros nueve países, si no muestre los siguientes 10
  let currentCountries;
  currentPage === 1
    ? currentCountries = countries.slice(0, 9)
    : currentCountries = countries.slice(firstIndex-1, lastIndex-1); //slice(9, 19) = 9, 10, 11,...18

  ⚠ const pagination = (pageNumber) => {
    setCurrentPage(pageNumber);
  }

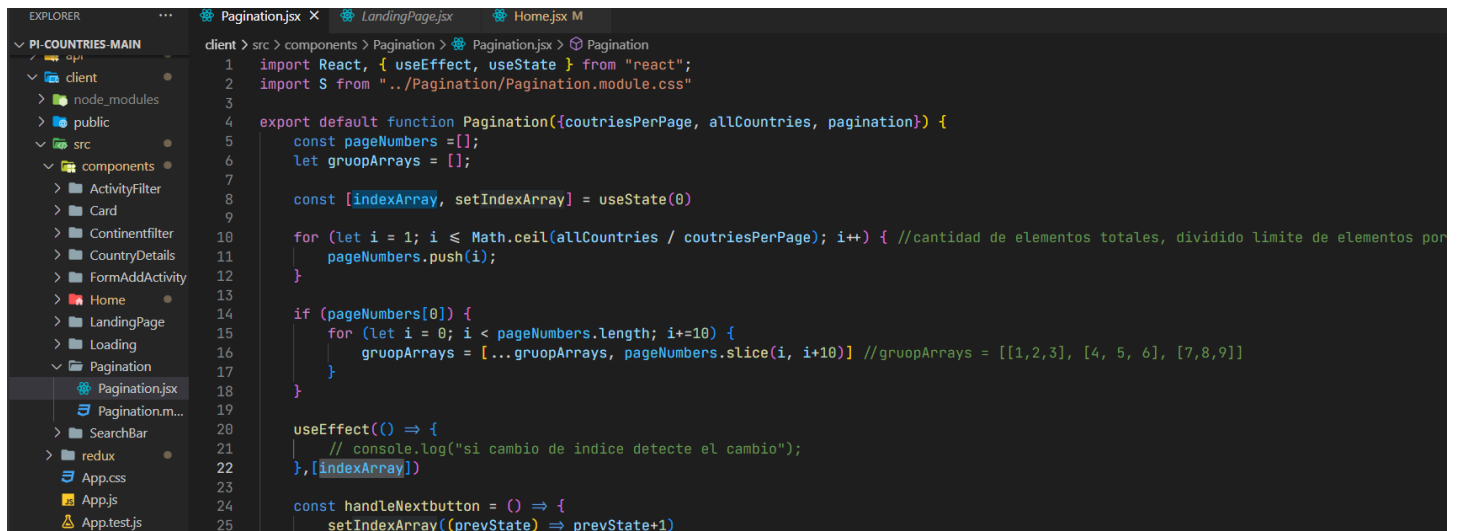
  //al cambiar el contenido del arreglo countries se establezca que inicie en la primer pagina
  useEffect(() => {
    setCurrentPage(1);
  }, [countries])
}
```

Uso de useSelector() y useDispatch()

Para la paginación establecimos un estado de react para determinar en qué página estamos actualmente que inicialmente será 1, establecemos que queremos 10 elementos por página por defecto, lastIndex y firstIndex serán las variables encargadas de manejar que posiciones del arreglo se tomarán para ser mostradas en pantalla que irán de 10 en 10. Luego creamos una nueva variable llamada current countries donde irán los países según la página. Creamos un operador ternario donde diremos que si estamos en la página 1 muestre los primeros 9 elementos, de lo contrario mostrara 10 elementos, donde nos desplazaremos una posición atrás en los índices que habías establecido que serían múltiplos de 10.

Creamos una función llamada pagination que será pasada como parámetro al componente pagination, función la cual se encargara de setear el valor de la pagina en la que nos encontramos.

En la parte de abajo usamos un useEffect para decirle al componente que cada vez que se modifique el arreglo total de países me ubique en la página 1 de dichos elementos.



```
client > src > components > Pagination > Pagination.jsx > Pagination
1  import React, { useEffect, useState } from "react";
2  import S from "../Pagination/Pagination.module.css"
3
4  export default function Pagination({countriesPerPage, allCountries, pagination}) {
5    const pageNumbers = [];
6    let groupArrays = [];
7
8    const [indexArray, setIndexArray] = useState(0)
9
10   for (let i = 1; i <= Math.ceil(allCountries / countriesPerPage); i++) { //cantidad de elementos totales, dividido limite de elementos por
11     pageNumbers.push(i);
12   }
13
14   if (pageNumbers[0]) {
15     for (let i = 0; i < pageNumbers.length; i+=10) {
16       groupArrays = [...groupArrays, pageNumbers.slice(i, i+10)] //groupArrays = [[1,2,3], [4, 5, 6], [7,8,9]]
17     }
18   }
19
20   useEffect(() => {
21     // console.log("si cambio de indice detecte el cambio");
22     setIndexArray(0);
23   }, [indexArray])
24
25   const handleNextbutton = () => {
26     setIndexArray((prevState) => prevState+1)
27   }
28 }
```

En el componente pagination recibimos por parámetro cantidad de países por página, el tamaño del arreglo con todos los países y una función pagination encargada de setear el estado que especifica en que página estamos.

El arreglo pageNumbers almacena el total de páginas, en un mismo arreglo. Luego validamos si dicho arreglo no está vacío vamos a separarlos en subArreglos más pequeños que se agruparan dentro de este con límite de máximo 10 páginas en cada sub-arreglo. Dentro de if hay un for que se encarga de agregar los 10 elementos, luego aumenta el índice en 10 y pushea los siguientes 10

Tenemos un `useEffect` que se encarga de estar atento si nosotros decidimos cambiar de posición en el arreglo esto por medio de los botones `prev` y `next`.

```
23
24
25 const handleNextbutton = () => {
26   setIndexArray((prevState) => prevState+1)
27 };
28
29 const handleBeforeButton = () => {
30   setIndexArray((prevState) => prevState-1)
31 };
32
33 return(
34   <nav className={S.pagination_nav}>
35     { //si no está en el primer grupo de paginas no me deje devolver
36       grupoArrays[0] && indexArray === 0
37       ? <button className={S.button_go_pages} onClick={handleBeforeButton}><" ">prev</button>
38       : ""
39     }
40     <ul className={S.pagination_ul}>
41       {
42         grupoArrays[0] && grupoArrays[indexArray].map(number => (
43           <li onClick={() => pagination(number)} key={number} className={S.pagination_li}>
44             <button type="button">
45               {number}
46             </button>
47           </li>
48         ))
49       }
50     </ul>
51     { //si estoy en la ultima posicion de paginas no me muestre next
52       grupoArrays[0] && indexArray === grupoArrays.length-1
53       ? <button className={S.button_go_pages} onClick={handleNextbutton}>next{" "></button>
54       : ""
55     }
56   </nav>
57 )
```

Tenemos las funciones `handleNext` y `handlebefore`, que nos mueven de una posición del arreglo de grupos a otra, luego especificamos que si hay elementos dentro el arreglo de grupos y la posición es diferente del valor 0 me renderice el botón de `prev` (ir atrás).

Renderiza las paginas y al hacer clic en algún numero este mostrara los países pertenecientes a esa página.

Luego usamos un ternario para preguntar si estamos en una pagina diferente a la ultima y si es el caso me renderice el botón de `next` (ir al siguiente grupo de páginas).

HOME

```
45
46
47 useEffect(() => {
48   setCurrentPage(1);
49 }, [countries])
50
51 useEffect(() => {
52   dispatch(getAllCountries());
53   dispatch(getAllActivities());
54 }, [dispatch]);
55
56 const handleOrderByName = (e) => {
57   e.preventDefault();
58   dispatch(orderByName(e.target.value));
59   setUpdate(e.target.value);
60 };
61
62 const handleOrderByPopulation = (e) => {
63   e.preventDefault();
64   dispatch(orderByPopulation(e.target.value));
65   setUpdate(e.target.value);
66 };
67
68 document.title = "Home";
69 return(
```

Usamos un `useEffect` component `didMount` para que al cargarse el componente me me traiga las actividades y los países, también un component `diUpdate` para que se actualicen los valores esos arreglos luego de cada `dispatch`, los `dispatch` se ejecutaran en las funciones especificadas cada vez que estas sean llamadas.

```

75     <div className={S.container_left}>
76       <SearchBar />
77       <div className={S.container_filters}>
78         <select onChange={handleOrderByName}>
79           <option disabled defaultVal selected>Alphabetical</option>
80           <option value="A-Z">A - Z</option>
81           <option value="Z-A">Z - A</option>
82         </select>
83
84         <select onChange={handleOrderByPopulation}>
85           <option disabled defaultVal selected>Population</option>
86           <option value="max-p">max population</option>
87           <option value="min-p">min population</option>
88         </select>
89         .....
90       <ContinentFilter />
91
92       <ActivityFilter allActivities={allActivities} />
93     </div>
94   </div>

```

En la parte superior de la pagina tendremos estos elementos que nos permitirán organizar y filtrar los países según ciertas características de ellos. Tomamos dichas características y hacemos un dispatch pasado como parámetro el valor con el que deseamos filtrar u organizar.

```

103 > node_modules 103
104 > public 104
105 > src 105
106 > components 106
107 > ActivityFilter 107
108 > Card 108
109 > Continentfilter 109
110 > CountryDetails 110
111 > FormAddActivity 111
112 > Home 112
113 > images 113
114 > Home.jsx M 114
115 > Home.module... 115
116 > LandingPage 116
117 > images 117
118 > LandingPage.j... 118
119 > LandingPage... 119
120 > Loading 120
121 > Pagination 121
122 > Pagination.jsx 122
123 > Pagination.m... 123
124 > Searchbar 124
125 > redux 125
126 > App.css 126
127 > App.js 127
128 > App.test.js 128
129 > index.css 129
130 > index.js 130
131 > reportWebVitals.js 131
132 > ... 132
133 > ... 133
134 > ... 134
135 > ... 135

```

```

103 {
104   !currentCountries[0]
105   ? <Loading />
106   :
107   <div className={S.cards_pagination_container}>
108     <div className={S.countries_container}>
109       {
110         currentCountries.map((el) => {
111           return(
112             el.id === "error404" ? //si no llega a existir el pais buscado
113             <div key={el.id}>
114               <Link to={'/home'} onClick={() => window.location.reload(false)}>
115                 {
116                   <Card key={el.id} imgFlag={el.img_flag} nameCountry={el.name} continent={el.continent}/>
117                 }
118               </Link>
119             </div>
120             :
121             <div key={el.id}>
122               <Link to={'/country-detail/${el.id}'}>
123                 {
124                   <Card key={el.id} imgFlag={el.img_flag} nameCountry={el.name} continent={el.continent}/>
125                 }
126               </Link>
127             </div>
128           )
129         })
130       }
131     </div>
132   <Pagination className={S.pagination_container} countriesPerPage={countriesPerPage} allCountries={countries.length} pagin
133   </div>
134 }
135 }

```

En caso de que el arreglo de países este vacío muestre un componente de cargando, si no renderice los países traídos, en caso de que el país traído tenga un id de 404error me renderice una tarjeta con su data y un link que al darle clic recargue todo el sitio.

Finalmente, en la parte inferior tendrá l componente de paginado al cual le pasaremos por props cantidad de países por página, longitud total del arreglo de países y una función llamada pagination.

CARD

```

Card.jsx
client > src > components > Card > Card.jsx > Card
1  import React from "react";
2  import S from "../Card/Card.module.css";
3
4  export default function Card({ imgFlag, nameCountry, continent }) {
5      return(
6          <div className={S.card_container}>
7              <div className={S.image_container}>
8                  <img src={imgFlag} alt={` ${nameCountry} `} className={S.flag_image}/>
9              </div>
10             <div className={S.text_container}>
11                 <h2>{nameCountry}</h2>
12                 <h2>{continent}</h2>
13             </div>
14         </div>
15     )
16 }

```

componente que mostrara una tarjeta, con los valores del link de imagen de la bandera, nombre y continente. Valores que recibirá por props

SEARCH BAR

```

Home.jsx M  SearchBar.jsx M
client > src > components > SearchBar > SearchBar.jsx > SearchBar
1  import React, { useState } from "react";
2  import { useDispatch } from "react-redux";
3  import { getCountryByName } from "../../redux/actions"
4  import S from "../SearchBar/SearchBar.module.css";
5
6  export default function SearchBar() {
7      const dispatch = useDispatch();
8      const [nameCountry, setNameCountry] = useState("");
9
10     const handleInput = (e) => {
11         e.preventDefault();
12         setNameCountry(e.target.value);
13     }
14
15     const handleSubmit = (e) => {
16         e.preventDefault();
17         dispatch(getCountryByName(nameCountry));
18     }
19
20     return(
21         <div className={S.search_container}>
22             <input type="text" className={S.input_search} onChange={handleInput} placeholder="Name of country..." />
23             <button className={S.buton_search} onClick={handleSubmit}>
24                 Search
25             </button>
26         </div>
27     )
28 }

```

Este componente recibirá un valor en el input y al hacer clic en el botón de búsqueda despachará la función getCountrybyName().

CONTINENT FILTER

```
EXPLORER
PI-COUNTRIES-MAIN
> api
> client
  > node_modules
  > public
  > src
    > components
      > Continentfilter
        > Continentfilter.jsx
        > Continentfilter.mod...
      > CountryDetails
      > FormAddActivity
      > Home
      > LandingPage
      > Loading
      > Pagination
      > SearchBar
    > redux
      > App.css
      > App.js
      > App.test.js
      > index.css
      > index.js
      > reportWebVitals.js
      > setupTests.js
    > env

client > src > components > Continentfilter > Continentfilter.jsx > ...
1  import React from "react";
2  // import { useState } from "react";
3  import { useDispatch } from "react-redux";
4  import { filterByContinent } from "../../redux/actions";
5
6  export default function ContinentFilter() {
7
8      const dispatch = useDispatch();
9
10     const allContinents = ["Africa", "Antarctica", "Asia", "Europe", "Oceania", "North America", "South America"];
11
12     const handleFilterByContinent = (e) => {
13         e.preventDefault();
14         dispatch(filterByContinent(e.target.value));
15         console.log(e.target.value);
16     }
17
18     return (
19         <select onChange={handleFilterByContinent}>
20             <option value="" disabled defaultValue>
21                 Continents
22             </option>
23             <option value="All">All continents</option>
24             {
25                 allContinents.map(continent => (
26                     <option key={continent} value={continent}>{continent}</option>
27                 ))
28             }
29         </select>
30     );
31 }
```

Tendremos un arreglo con todos los continentes.

Mapearemos esos valores del arreglo en un select y al seleccionar alguno de esas opciones en el select despachará la función `filterByContinent()` y como parámetro le pasará el continente seleccionado que su valor estará en su atributo `html value=""`. Tendremos un valor extra en el select llamado “all continents”, el cual traerá los países de todos los continentes de vuelta.

ACTIVITY FILTER

```
EXPLORER
PI-COUNTRIES-MAIN
> api
> client
  > node_modules
  > public
  > src
    > components
      > ActivityFilter
        > ActivityFilter.jsx
        > ActivityFilter.modul...
      > Card
      > Continentfilter
      > CountryDetails
      > FormAddActivity
      > Home
      > LandingPage
      > Loading
      > Pagination
      > SearchBar
    > redux
      > App.css
      > App.js
      > App.test.js
      > index.css
      > index.js

client > src > components > ActivityFilter > ActivityFilter.jsx > ActivityFilter
1  import React from "react";
2  import { useDispatch } from "react-redux";
3  import { filterByActivity } from "../../redux/actions";
4
5  export default function ActivityFilter({allActivities}) {
6
7      const dispatch = useDispatch();
8
9      const handleFilterByActivity = (e) => {
10         e.preventDefault();
11         dispatch(filterByActivity(e.target.value));
12         console.log(e.target.value);
13     }
14
15     return(
16         <select onChange={handleFilterByActivity}>
17             <option value="" disabled defaultValue selected>
18                 Activities
19             </option>
20             <option value="All">all countries</option>
21             {
22                 allActivities.map(act =>
23                     <option key={act.name} value={act.id}>{act.name}</option>
24                 )
25             }
26         </select>
27     );
28 }
```

Por props recibimos un arreglo con todas las actividades registradas en la base de datos, mapearemos dicho arreglo y lo renderizamos como las options de un select, al hacer clic sobre alguna de las opciones este ejecuta una función que dispatcha la acción `filterByActivity()` pasandole como parámetro el id de dicha acción.

DETAILS

```
> api
  client
    node_modules
    public
    src
      components
        ActivityFilter
        Card
        Continentfilter
        CountryDetails
        CountryDetails.jsx
        CountryDetails.mod...
        FormAddActivity
        Home
        LandingPage
        Loading
        Pagination
        SearchBar
        redux
        App.css
1  import React, { useEffect } from "react";
2  import { useDispatch, useSelector } from "react-redux";
3  import { useParams } from "react-router-dom";
4  import { getByID, resetDetailById } from "../../redux/actions";
5  import { Link } from "react-router-dom";
6  import S from "../../CountryDetails/CountryDetails.module.css"
7  import Loading from "../../Loading/Loading";
8
9  export default function CountryDetails() {
10    const dispatch = useDispatch();
11    let { id } = useParams();
12
13    let details = useSelector((state) => state.details);
14
15    //trae los detalles segun el id del pais
16    useEffect(() => {
17      dispatch(getByID(id));
18    }, [dispatch, id]);
19
20    const handleReset = () => {
21      dispatch(resetDetailById());
22    }
23  }
```

importamos el hook useParams el cual nos permite leer la parte del enlace pasada como params, tomar ese valor y almacenarlo en una variable.

Usamos use selector para traer los detalles del país según su id, id que será tomado de params. Usaremos component didMount para que al montar el componente este despache la acción getByld() pasandole como parámetro el id tomado de params. A su vez será un component didUpdate que estará atento a las modificaciones del id y dispatch por si se selecciona un país con un id diferente.

Tenemos una función de reset que al dar click en el Link de home para volver al componente Home despacha la acción resetDetailById() que se encargara de vaciar los valores del estado de detalles.

A su vez el valor del arreglo detalles contendrá las actividades vinculadas con el país las cuales se mapearán y se renderizarán en la parte derecha del de la tarjeta del país.

FORM ADD ACTIVITY

```
PI-COUNTRIES-... client > src > components > FormAddActivity > FormAddActivity.jsx > FormAddActivity
  client
    node_modules
    public
    src
      components
        ActivityFilter
        Card
        Continentfilter
        CountryDetails
        FormAddActivity
        FormAddActivity.jsx
        FormAddActivity.m...
        Home
        LandingPage
        Loading
        Pagination
        SearchBar
        redux
        App.css
        App.js
        App.test.js
1  import React from "react";
2  import { useState, useEffect } from "react";
3  import { useDispatch, useSelector } from "react-redux";
4  import { Link } from "react-router-dom";
5  import { getAllCountries, postActivity } from "../../redux/actions";
6  import S from "../../FormAddActivity/FormAddActivity.module.css";
7  import Loading from "../../Loading/Loading";
8
9  export default function FormAddActivity() {
10    const dispatch = useDispatch();
11
12    const [formValues, setFormValues] = useState({
13      name: "",
14      difficulty: "",
15      duration: "",
16      season: "",
17      countries_id: [],
18    });
19    // al cargar el componente muestra los mensajes de error
20    const [errorForm, setErrorForm] = useState({
21      name: "this field is required",
22      countries: "select at least one country",
23    });
24
25    const [enableButt, setEnableButt] = useState(true);
```

Creamos un estado de react en el cual almacenaremos los valores ingresados en cada campo del formulario, y creamos otro estado de react para almacenar los mensajes de error que mostraremos los cuales serán dos ya que solo dejaremos dos campos obligatorios. Y al final otro estado en el cual habilitaremos o deshabilitaremos el botón de enviar formulario en caso de que no haya ningún error.

```
EXPLORER
client > src > components > FormAddActivity > FormAddActivity.jsx > FormAddActivity > useEffect() callback

29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55

useEffect(() => {
  if (errorForm.name.length > 0 || errorForm.countries.length > 0) {
    setEnableButt(true)
  }
  else{
    setEnableButt(false);
  }
}, [errorForm])

useEffect(() => {
  dispatch(getAllCountries());
}, [dispatch]);

useEffect(() => {
  if (formValues.countries_id.length < 1) {
    setErrorForm({
      ...errorForm,
      countries: "select at least one country"
    })
  }if (formValues.countries_id.length > 0) {
    setErrorForm({
      ...errorForm,
      countries: ""
    })
  }
// eslint-disable-next-line react-hooks/exhaustive-deps
}, [formValues.countries_id])
```

Usaremos tres useEffect, el primero se encargará de estar atento a los cambios de los valores del estado de errores del formulario. Validamos que no exista ningún error en los valores del errorForm y si es el caso habilita el botón de enviar.

El segundo useEffect() se encarga de cargar todos los países.

Y el tercer useEffect valida si se ha elegido al menos un país de la lista, y en caso contrario de no hacerlo modifica el mensaje de error.

```
57 const hours = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12];
58
59 const validationform = (e) => {
60   // validar que el campo de nombre sean letras y no este vacio
61   if (e.target.name === "name") {
62     if (e.target.value === "") {
63       return(
64         setErrorForm({
65           ...errorForm,
66           name: "this field is required",
67         })
68       )
69     }
70     if (!/^[A-Z]+$/.test(e.target.value)) {
71       return(
72         setErrorForm({
73           ...errorForm,
74           name: "only letters are allowed"
75         })
76       )
77     }
78     if (/^[A-Z]+$/.test(e.target.value)) {
79       return(
80         setErrorForm({
81           ...errorForm,
82           name: ""
83         })
84       )
85     }
86   }
```

Creamos un arreglo con todas las opciones de duración, para luego mapearlas y mostrarlas en el select.

Creamos una función que valida que sean una modificación en el campo name, donde pondremos el nombre de la actividad y arroja un error o no según los valores ingresados, donde usaremos una expresión regular.


```

88   const handleChange = (e) => {
89       setFormValues({
90           ...formValues,
91           [e.target.name]: e.target.value
92       })
93       validationform(e);
94   }
95   //seleccionar pais
96   const handleSelectCount = (e) => {
97       setFormValues({
98           ...formValues,
99           countries_id: [...formValues.countries_id, e.target.value]
100       })
101       validationform(e);
102   }

```

Tenemos dos funciones la cual `handleChange` se encargará de setear los valores en el estado de `formValues`, esto según el atributo `value` del campo elegido. Y a su vez llamaremos la función `validation` y le pasaremos como parámetro la data del evento con lo cual ella validará si hay algún error o no en ello.

En la función `handleSelectCount()` tomamos el id del país seleccionado y lo agregamos a al arreglo.

```

104   const handleDelete = (countryDelete) => {
105       setFormValues({
106           ...formValues,
107           countries_id: formValues.countries_id.filter(count => count !== countryDelete)
108       })
109   }
110
111   const handleSubmit = (e) => {
112       e.preventDefault();
113       dispatch(postActivity(formValues));
114       alert("The new activity was added succesfully");
115       setFormValues({
116           name: "",
117           difficulty: "",
118           duration: "",
119           season: "",
120           countries_id: [],
121       });
122       //reset del formulario
123       e.target[1].value = ""; //resetear difficulty
124       e.target[2].value = ""; //resetear duration
125       e.target[3].value = ""; //resetear season
126       e.target[4].value = ""; //resetear countries
127   }
128
129   const allCountries = useSelector((state) => state.totalData)

```

En la función `handleDelete()` recibimos por parámetro el id del país sobre el que hicimos clic y hacemos un filtrado para ignorar ese elemento en el arreglo y de esta manera eliminarlo del arreglo `countries_id` del estado de react de países a vincular con dicha actividad.

En la función `handleSubmit()` despachamos la acción `postActivity()` pasándole por parámetro los valores a enviar por body que se encuentra en estado llamado `formValues` y que se inserten en la base de datos según se estableció en dicha ruta, a su vez hacemos un reset de los valores en el estado para que luego de que se ejecute esa acción el formulario limpie los campos del formulario.

Hacemos un `useSelector` para traer el arreglo con todos los países en ese estado, almacenado en el store el cual no se modifica en absoluto.