**React Redux Webpack практический курс**

**1 БАЗОВЫЕ НАСТРОЙКИ WEBPACK**

**npm init**

Далее задаем натсройки

Создает package.json

**Подключаем webpack**

**npm install –D webpack webpack-cli**

**webpack-cli** пакет который позволяет использовать webpack через командную строку

**создаем папку src в ней index.js and index.html**

**создаем файл webpack.confog.js для конфигурации вебпака**

/\* Импортируем из node.js библиотеку path которая позволит нам указать путь к папке \*/

const path = require("path");

module.exports = {

  /\* mode - указывает что находимсяв  разработке и выходные данные JS не надо сжимать

  entry - затем надо указать путь к файлу который будет входным с которого будет начинаться запуск приложения

  output - куда будет сам вебпак собирать файлы

  \*/

  mode: "development",

  entry: "./src/index.js",

  output: {

    /\* path - путь куда webpack будет делать сборку папка dist \*/

    path: path.resolve(\_\_dirname, "dist"),

    /\* Имя файла куда вебпак будет делать сборку всех файлов \*/

    /\* Будем использовать регулярные выражения чтобы не было проблем при хэшировании разработки чтобы он делал бандлы с уникальными названиями\*/

    filename: "[name].[hash].js",

  },

};

Для того чтобы писать для каждого компоненты стили их нужно импортировать

**npm install –D style-loader less-loader css-loader file-loader**

less –будем использовать препроцессро less

css - css

file – позволяет импортировать файлы различные изображения и тд

Устанавливаем плагины для webpack он предоставляет много различных но мы вопсользуемся двумя

**npm install –D html-webpack-plugin clean-webpack-plugin**

**или**

**npm install html-webpack-plugin --save-dev**

**npm install clean-webpack-plugin --save-dev**

html-webpack-plugin – переносит наш шаблон в папку dist и ипортирует в него все соответствующие JS бандлы которые будет собирать webpack

clean-webpack-plugin – очищает файлы не используемые в filename регулярных выражений

После того как модули скачались покдлючаем их и называем с большой буквы так как это классы

/\* Импортируем из node.js библиотеку path которая позволит нам указать путь к папке \*/

const path = require("path");

**const { CleanWebpackPlugin } = require("clean-webpack-plugin");**

**const HTMLWebpackPlugin = require("html-webpack-plugin");**

module.exports = {

  /\* mode - указывает что находимсяв  разработке и выходные данные JS не надо сжимать

  entry - затем надо указать путь к файлу который будет входным с которого будет начинаться запуск приложения

  output - куда будет сам вебпак собирать файлы

  \*/

  mode: "development",

  entry: "./src/index.js",

  output: {

    /\* path - путь куда webpack будет делать сборку папка dist \*/

    path: path.resolve(\_\_dirname, "dist"),

    /\* Имя файла куда вебпак будет делать сборку всех файлов \*/

    /\* Будем использовать регулярные выражения чтобы не было проблем при хэшировании разработки чтобы он делал бандлы с уникальными названиями\*/

    filename: "[name].[hash].js",

  },

**/\* Теперь подключаем плагины указываем путь к HTML \*/**

**plugins: [**

**new HTMLWebpackPlugin({ template: "./src/index.html" }),**

**new CleanWebpackPlugin(),**

  ],

};

Вставили эти две строчки

{

  "name": "react",

  "version": "1.0.0",

  "description": "",

  "main": "index.js",

  "scripts": {

    "test": "echo \"Error: no test specified\" && exit 1",

    "dev": "webpack --mode development",

    "build": "webpack --mode production"

  },

  "author": "",

  "license": "ISC",

  "devDependencies": {

    "clean-webpack-plugin": "^4.0.0",

    "css-loader": "^6.7.3",

    "file-loader": "^6.2.0",

    "html-webpack-plugin": "^5.5.0",

    "less-loader": "^11.1.0",

    "style-loader": "^3.3.2",

    "webpack": "^5.77.0",

    "webpack-cli": "^5.0.1"

  }

}

Потом собираем вебпак

**npm run dev**

**Создаётся папка dist вебпаком в неё был перенесен наш index.html а в него импортирован JS бандел**

Для того чтобы изменения не пересобирать нужно утсановить который будет выполнять эти действия сам

**npm install –D webpack-dev-server**

/\* Импортируем из node.js библиотеку path которая позволит нам указать путь к папке \*/

const path = require("path");

const { CleanWebpackPlugin } = require("clean-webpack-plugin");

const HTMLWebpackPlugin = require("html-webpack-plugin");

module.exports = {

  /\* mode - указывает что находимсяв  разработке и выходные данные JS не надо сжимать

  entry - затем надо указать путь к файлу который будет входным с которого будет начинаться запуск приложения

  output - куда будет сам вебпак собирать файлы

  \*/

  mode: "development",

  entry: "./src/index.js",

  output: {

    /\* path - путь куда webpack будет делать сборку папка dist \*/

    path: path.resolve(\_\_dirname, "dist"),

    /\* Имя файла куда вебпак будет делать сборку всех файлов \*/

    /\* Будем использовать регулярные выражения чтобы не было проблем при хэшировании разработки чтобы он делал бандлы с уникальными названиями\*/

    filename: "[name].[hash].js",

  },

  /\* Теперь подключаем плагины указываем путь к HTML \*/

  plugins: [

    new HTMLWebpackPlugin({ template: "./src/index.html" }),

    new CleanWebpackPlugin(),

  ],

**/\* Настраиваем ипорт наших стилей**

**для этого задаем определенные правила \*/**

**module: {**

**rules: [**

**{**

**/\* Будет обрабатывать указанные ниже loader css и less файлы\*/**

**test: /\.(css|less)$/,**

**/\* Указываем массив loader который будет использоваться с лева на право учитываем \*/**

**use: ["style-loader", "css-loader", "less-loader"],**

**},**

**],**

**},**

**};**

Создаем index.less

h1 {

  color: red;

}

Подключаем в index.js

Пересобираем вебпак и проверяем что всё работает

**npm run dev**

/\* Импортируем из node.js библиотеку path которая позволит нам указать путь к папке \*/

const path = require("path");

const { CleanWebpackPlugin } = require("clean-webpack-plugin");

const HTMLWebpackPlugin = require("html-webpack-plugin");

module.exports = {

  /\* mode - указывает что находимсяв  разработке и выходные данные JS не надо сжимать

  entry - затем надо указать путь к файлу который будет входным с которого будет начинаться запуск приложения

  output - куда будет сам вебпак собирать файлы

  \*/

  mode: "development",

  entry: "./src/index.js",

  output: {

    /\* path - путь куда webpack будет делать сборку папка dist \*/

    path: path.resolve(\_\_dirname, "dist"),

    /\* Имя файла куда вебпак будет делать сборку всех файлов \*/

    /\* Будем использовать регулярные выражения чтобы не было проблем при хэшировании разработки чтобы он делал бандлы с уникальными названиями\*/

    filename: "[name].[hash].js",

  },

**/\* Настраиваем devServer \*/**

**devServer: {**

**port: 3000,**

**},**

  /\* Теперь подключаем плагины указываем путь к HTML \*/

  plugins: [

    new HTMLWebpackPlugin({ template: "./src/index.html" }),

    new CleanWebpackPlugin(),

  ],

  /\* Настраиваем ипорт наших стилей

  для этого задаем определенные правила \*/

  module: {

    rules: [

      {

        /\* Будет обрабатывать указанные ниже loader css и less файлы\*/

        test: /\.(css|less)$/,

        /\* Указываем массив loader который будет использоваться с лева на право учитываем \*/

        use: ["style-loader", "css-loader", "less-loader"],

      },

**{**

**/\* Делаем обработку для файлов \*/**

**test: /\.(jpg|jpeg|png|svg)$/,**

**use: ["file-loader"],**

**},**

    ],

  },

};

**Флаг open позволит открывать приложения сразу после запуска скрипта**

{

  "name": "react",

  "version": "1.0.0",

  "description": "",

  "main": "index.js",

  "scripts": {

    "test": "echo \"Error: no test specified\" && exit 1",

    "dev": "webpack --mode development",

    "build": "webpack --mode production",

**"start": "webpack-dev-server --open"**

  },

  "author": "",

  "license": "ISC",

  "devDependencies": {

    "clean-webpack-plugin": "^4.0.0",

    "css-loader": "^6.7.3",

    "file-loader": "^6.2.0",

    "html-webpack-plugin": "^5.5.0",

    "less-loader": "^11.1.0",

    "style-loader": "^3.3.2",

    "webpack": "^5.77.0",

    "webpack-cli": "^5.0.1",

    "webpack-dev-server": "^4.13.2"

  }

}

И теперь можем запускать приложение через

**npm start**

**3. React Redux - настраиваем React в Webpack**

Необходимо подключить Bable преобразует код чтобы все браузеры могли его понимать а также преобразует разметку JSX в JS

Перешли на сайт Babel и в setup нашли webpack

**npm install --save-dev babel-loader @babel/core**

**Копируем от туда же правила и копируем в webpack**

{  
 module: {  
 rules: [  
 {  
 test: /\.m?js$/,  
 exclude: /node\_modules/,  
 use: {  
 loader: "babel-loader",  
 options: {  
 presets: ['@babel/preset-env']  
 }  
 }  
 }  
 ]  
 }  
}

**Также устанавливаем пресет который будет преобразовывать код для старых браузеров**

**npm install @babel/preset-env --save-dev**

**Также устанавливаем пресет чтобы он понимал реакт**

**npm install --save-dev @babel/preset-react**

{

        /\* Скопировали добавили jsx и пресет для реакта\*/

        test: /\.m?jsx$/,

        exclude: /node\_modules/,

        use: {

          loader: "babel-loader",

          options: {

            presets: ["@babel/preset-env", "@babel/preset-react"],

          },

        },

      },

/\* Импортируем из node.js библиотеку path которая позволит нам указать путь к папке \*/

const path = require("path");

const { CleanWebpackPlugin } = require("clean-webpack-plugin");

const HTMLWebpackPlugin = require("html-webpack-plugin");

module.exports = {

  /\* mode - указывает что находимсяв  разработке и выходные данные JS не надо сжимать

  entry - затем надо указать путь к файлу который будет входным с которого будет начинаться запуск приложения

  output - куда будет сам вебпак собирать файлы

  \*/

  mode: "development",

  entry: ["./src/index.jsx"],

  output: {

    /\* path - путь куда webpack будет делать сборку папка dist \*/

    path: path.resolve(\_\_dirname, "dist"),

    /\* Имя файла куда вебпак будет делать сборку всех файлов \*/

    /\* Будем использовать регулярные выражения чтобы не было проблем при хэшировании разработки чтобы он делал бандлы с уникальными названиями\*/

    filename: "[name].[hash].js",

  },

  /\* Настраиваем devServer \*/

  devServer: {

    port: 3000,

  },

  /\* Теперь подключаем плагины указываем путь к HTML \*/

  plugins: [

    new HTMLWebpackPlugin({ template: "./src/index.html" }),

    new CleanWebpackPlugin(),

  ],

  /\* Настраиваем ипорт наших стилей

  для этого задаем определенные правила \*/

  module: {

    rules: [

      {

        /\* Будет обрабатывать указанные ниже loader css и less файлы\*/

        test: /\.(css|less)$/,

        /\* Указываем массив loader который будет использоваться с лева на право учитываем \*/

        use: ["style-loader", "css-loader", "less-loader"],

      },

      {

        /\* Делаем обработку для файлов \*/

        test: /\.(jpg|jpeg|png|svg)$/,

        use: ["file-loader"],

      },

      {

        /\* Правила из Babel \*/

        test: /\.m?js$/,

        exclude: /node\_modules/,

        use: {

          loader: "babel-loader",

          options: {

            presets: ["@babel/preset-env"],

          },

        },

      },

      {

        /\* Скопировали добавили jsx и пресет для реакта\*/

        test: /\.m?jsx$/,

        exclude: /node\_modules/,

        use: {

          loader: "babel-loader",

          options: {

            presets: ["@babel/preset-env", "@babel/preset-react"],

          },

        },

      },

    ],

  },

};

**Чтобы Babel работал правильно надо добавить полифил**

**npm install --save @babel/polyfill**

/\* Импортируем из node.js библиотеку path которая позволит нам указать путь к папке \*/

const path = require("path");

const { CleanWebpackPlugin } = require("clean-webpack-plugin");

const HTMLWebpackPlugin = require("html-webpack-plugin");

module.exports = {

  /\* mode - указывает что находимсяв  разработке и выходные данные JS не надо сжимать

  entry - затем надо указать путь к файлу который будет входным с которого будет начинаться запуск приложения

  output - куда будет сам вебпак собирать файлы

  \*/

  mode: "development",

  entry: ["@babel/polyfill", "./src/index.jsx"],

  output: {

    /\* path - путь куда webpack будет делать сборку папка dist \*/

    path: path.resolve(\_\_dirname, "dist"),

    /\* Имя файла куда вебпак будет делать сборку всех файлов \*/

    /\* Будем использовать регулярные выражения чтобы не было проблем при хэшировании разработки чтобы он делал бандлы с уникальными названиями\*/

    filename: "[name].[hash].js",

  },

  /\* Настраиваем devServer \*/

  devServer: {

    port: 3000,

  },

  /\* Теперь подключаем плагины указываем путь к HTML \*/

  plugins: [

    new HTMLWebpackPlugin({ template: "./src/index.html" }),

    new CleanWebpackPlugin(),

  ],

  /\* Настраиваем ипорт наших стилей

  для этого задаем определенные правила \*/

  module: {

    rules: [

      {

        /\* Будет обрабатывать указанные ниже loader css и less файлы\*/

        test: /\.(css|less)$/,

        /\* Указываем массив loader который будет использоваться с лева на право учитываем \*/

        use: ["style-loader", "css-loader", "less-loader"],

      },

      {

        /\* Делаем обработку для файлов \*/

        test: /\.(jpg|jpeg|png|svg)$/,

        use: ["file-loader"],

      },

      {

        /\* Правила из Babel \*/

        test: /\.m?js$/,

        exclude: /node\_modules/,

        use: {

          loader: "babel-loader",

          options: {

            presets: ["@babel/preset-env"],

          },

        },

      },

      {

        /\* Скопировали добавили jsx и пресет для реакта\*/

        test: /\.m?jsx$/,

        exclude: /node\_modules/,

        use: {

          loader: "babel-loader",

          options: {

            presets: ["@babel/preset-env", "@babel/preset-react"],

          },

        },

      },

    ],

  },

};

**Меняет расширение index.js на jsx**

**Создаем папку components**

**npm install react react-dom**

import "./index.less";

import React from "react";

import { render } from "react-dom";

import App from "./components/App.jsx";

render(<App />, document.getElementById("root"));

**Создаем App и подключаем**

import React from "react";

function App() {

  return <div>React is working </div>;

}

export default App;

**Создали стили для компоненты APP**

import React from "react";

import "./app.less";

function App() {

  return <div className="app">React is working </div>;

}

export default App;

.app {

  color: red;

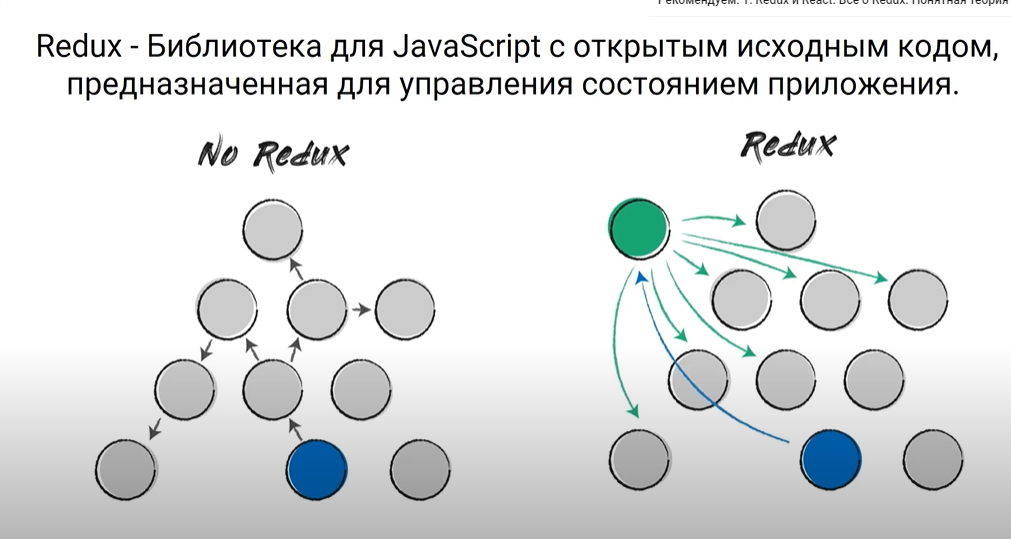
  font-size: 30px;

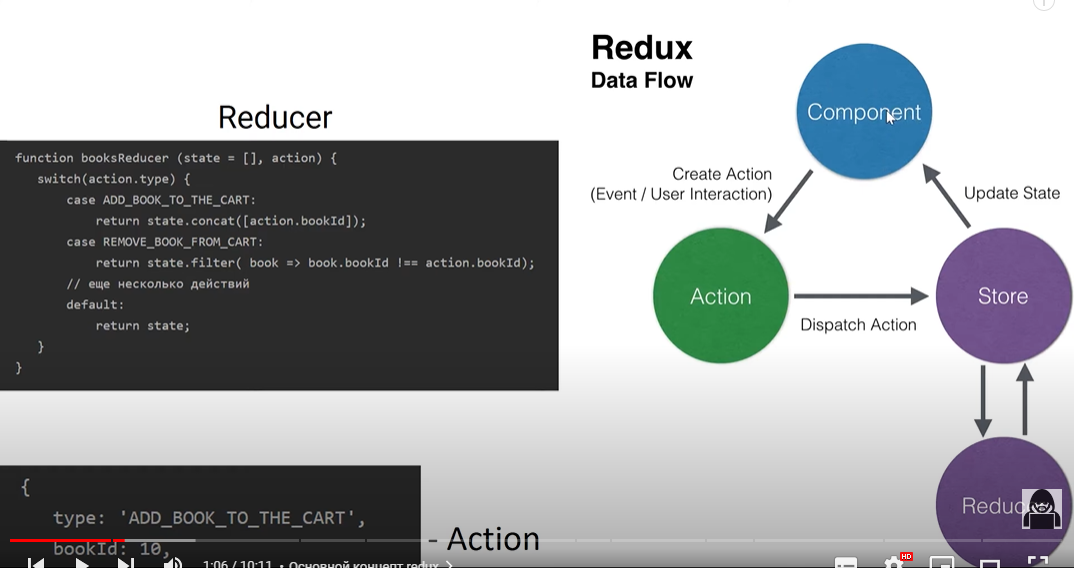
}

**4. React Redux - основы Redux, используем хуки useDispatch() и useSelector()**

**Управляет и хранит состояние приложения**

**Если большой проект то становится трудно управлять состоянием а в Redux оно находится в одном месте и можно передать его любым компонентам**

****

****

**Redux**

У нас есть компонента в которой мы можем создать action это простой JS объект который отслеживает события происходящие в объекте

Особенным action делает свойство type который является его необходимой частью

Action может содержать столько информации сколько мы хотим но лучшая практика когда он содержит мало информации

Как только определили action мы передаем его диспетчеру это функция из библиотеки которая принимает выполнения action и изменения состояния

Следующий этап это store представляет собой единый блок Который удерживает состояние приложения и методы с его взаимодейтсвием

И в конечном итоге нужна читая функция которая принимает state приложения и action и если надо меняет state и дает новый или старый если он не изменился это Reducer

Сделаем так чтобы каждый раз не писать jsx в компонентах

Переходим в официальную документацию webpack

/\* Импортируем из node.js библиотеку path которая позволит нам указать путь к папке \*/

const path = require("path");

const { CleanWebpackPlugin } = require("clean-webpack-plugin");

const HTMLWebpackPlugin = require("html-webpack-plugin");

module.exports = {

  /\* mode - указывает что находимсяв  разработке и выходные данные JS не надо сжимать

  entry - затем надо указать путь к файлу который будет входным с которого будет начинаться запуск приложения

  output - куда будет сам вебпак собирать файлы

  \*/

  mode: "development",

  entry: ["@babel/polyfill", "./src/index.jsx"],

  output: {

    /\* path - путь куда webpack будет делать сборку папка dist \*/

    path: path.resolve(\_\_dirname, "dist"),

    /\* Имя файла куда вебпак будет делать сборку всех файлов \*/

    /\* Будем использовать регулярные выражения чтобы не было проблем при хэшировании разработки чтобы он делал бандлы с уникальными названиями\*/

    filename: "[name].[hash].js",

  },

  /\* Настраиваем devServer \*/

  devServer: {

    port: 3000,

  },

  resolve: {

    extensions: [".js", ".jsx"],

  },

  /\* Теперь подключаем плагины указываем путь к HTML \*/

  plugins: [

    new HTMLWebpackPlugin({ template: "./src/index.html" }),

    new CleanWebpackPlugin(),

  ],

  /\* Настраиваем ипорт наших стилей

  для этого задаем определенные правила \*/

  module: {

    rules: [

      {

        /\* Будет обрабатывать указанные ниже loader css и less файлы\*/

        test: /\.(css|less)$/,

        /\* Указываем массив loader который будет использоваться с лева на право учитываем \*/

        use: ["style-loader", "css-loader", "less-loader"],

      },

      {

        /\* Делаем обработку для файлов \*/

        test: /\.(jpg|jpeg|png|svg)$/,

        use: ["file-loader"],

      },

      {

        /\* Правила из Babel \*/

        test: /\.m?js$/,

        exclude: /node\_modules/,

        use: {

          loader: "babel-loader",

          options: {

            presets: ["@babel/preset-env"],

          },

        },

      },

      {

        /\* Скопировали добавили jsx и пресет для реакта\*/

        test: /\.m?jsx$/,

        exclude: /node\_modules/,

        use: {

          loader: "babel-loader",

          options: {

            presets: ["@babel/preset-env", "@babel/preset-react"],

          },

        },

      },

    ],

  },

};

Делаем сборку и все работает

**npm run dev**

Теперь подключаем сам Redux

**npm i redux**

**npm i react-redux**

**npm i redux-thunk**

Создаем папку reducers

Index.js

И

reposReducer.js

В котором убдет храниться полученная с API гитхаба информация о репозиториях

/\* Объединяет все reducers которые есть в приложении \*/

import { combineReducers } from "redux";

/\* createStore позволяет создать Store а второй поключить redux-thunk  \*/

import { createStore, applyMiddleware } from "redux";

import reposReducer from "./reposReducer";

const rootReducer = combineReducers({

  repos: reposReducer,

});

/\* Создаем сам store \*/

const store = createStore(rootReducer, )

const defaultState = {

  /\* items полученные данные с github \*/

  items: [],

  /\* isFetching когда получим данные \*/

  isFetching: true,

};

export default function reposReducer(state = defaultState, action) {

  switch(action.type) {

    default:

    return state

  }

}

устанавливаем

**npm i redux-devtools-extension**

/\* Объединяет все reducers которые есть в приложении \*/

import { combineReducers } from "redux";

/\* createStore позволяет создать Store а второй поключить redux-thunk  \*/

import { createStore, applyMiddleware } from "redux";

import reposReducer from "./reposReducer";

/\* Отвечает за redux-devtools \*/

import { composeWithDevTools } from "redux-devtools-extension";

/\* redux thunk решает задачи с асинхронностью \*/

import thunk from "redux-thunk";

const rootReducer = combineReducers({

  repos: reposReducer,

});

/\* Создаем сам store \*/

export const store = createStore(

  rootReducer,

  composeWithDevTools(applyMiddleware(thunk))

);

import "./index.less";

import React from "react";

import { render } from "react-dom";

import App from "./components/App";

/\* Принимает store \*/

import { Provider } from "react-redux";

/\* Импортируем чтобы передать в Provider \*/

import { store } from "./components/reducers";

render(

  <Provider store={store}>

    <App />

  </Provider>,

  document.getElementById("root")

);

/\* У каждого action есть type и хорошая практика выносить всё это в отдельную переменную \*/

const SET\_COUNT = "SET\_COUNT";

const defaultState = {

  /\* items полученные данные с github \*/

  items: [],

  /\* isFetching когда получим данные \*/

  isFetching: true,

  count: 0,

};

export default function reposReducer(state = defaultState, action) {

  switch (action.type) {

    case SET\_COUNT:

      return {

        ...state,

        /\* count будет равен числу которое передадим в параметры \*/

        count: action.payload,

      };

    default:

      return state;

  }

}

/\* Также создадим Action creator это функция которая возращаем нам action

Это объект у которого есть поле type и какие либо данные

\*/

export const setCount = (count) => ({

  type: SET\_COUNT,

  payload: count,

});

import React from "react";

import "./app.less";

import { useDispatch, useSelector } from "react-redux";

import { setCount } from "./reducers/reposReducer";

function App() {

  /\* Берем из  useDispatch\*/

  const dispatch = useDispatch();

  /\*useSelector позволяет брать данные из state  \*/

  const count = useSelector((state) => state.repos.count);

  function onCountClick() {

    /\* Передаем в action \*/

    dispatch(setCount(5));

  }

  return (

    <div className="app">

      <button onClick={() => onCountClick()}>COUNT</button>

      <div>{count}</div>

    </div>

  );

}

export default App;

**5. React Redux - Главная страница. Axios - асинхронные запросы на API GitHub**

Устанавливаем

**npm i react-router-dom**

**npm i axios**

Создадим папку main где будут храниться все GitHub репозитории

Там создадим main.jsx and main.less для стилей

body {

  /\* Цинтруем всё на страницу \*/

  display: flex;

  justify-content: center;

}

import React from "react";

import "./app.less";

import { BrowserRouter, Route, Routes } from "react-router-dom";

function App() {

  return (

    <BrowserRouter>

      <Routes>

        <div className="container">

          <Route path="/" component={Main} />

        </div>

      </Routes>

    </BrowserRouter>

  );

}

export default App;

body {

  /\* Цинтруем всё на страницу \*/

  display: flex;

  justify-content: center;

}

/\* Делаем контейнер \*/

.container {

  width: 1080px;

}

Создаем папку actions где будут все запросы на сервер

import axios from "axios";

export const getRepos = (searchQuery = "stars:%3E1") => {

  /\* Так как используем redux-thunk нужно создать ещё одну функцию \*/

  return async (dispatch) => {

    const response = axios.get(

      `https://api.gihub.com/search/repositories?q=${searchQuery}&sort=stars`

    );

  };

};

/\* У каждого action есть type и хорошая практика выносить всё это в отдельную переменную \*/

const SET\_REPOS = "SET\_REPOS";

const defaultState = {

  /\* items полученные данные с github \*/

  items: [],

  /\* isFetching когда получим данные \*/

  isFetching: true,

};

export default function reposReducer(state = defaultState, action) {

  switch (action.type) {

    case SET\_REPOS:

      /\* Получает с API github репозитории \*/

      return {

        ...state,

        items: action.payload.items,

      };

    default:

      return state;

  }

}

/\* Также создадим Action creator это функция которая возращаем нам action

Это объект у которого есть поле type и какие либо данные

\*/

export const setRepos = (repos) => ({ type: SET\_REPOS, payload: repos });

Создаем папку для компонента repo он будет представлять из себя отдельно взятый репозиторий

import React from "react";

import { createRoot } from "react-dom/client";

/\* Принимает store \*/

import { Provider } from "react-redux";

/\* Импортируем чтобы передать в Provider \*/

import { store } from "./components/reducers";

import App from "./components/App";

createRoot(document.getElementById("root")).render(

  <Provider store={store}>

    <App />

  </Provider>

);

import React from "react";

import "./app.less";

import { BrowserRouter, Route, Routes } from "react-router-dom";

import Main from "./main/Main";

function App() {

  return (

    <BrowserRouter>

      <div className="container">

        <Routes>

          <Route path="/" element={<Main />} />

        </Routes>

      </div>

    </BrowserRouter>

  );

}

export default App;

import React, { useEffect } from "react";

import "./main.less";

import { useDispatch, useSelector } from "react-redux";

import { getRepos } from "../actions/repos";

import Repo from "./repo/repo";

function Main() {

  const dispatch = useDispatch();

  const repos = useSelector((state) => state.repos.items);

  /\* Если [] зависимости пустой то вызовится один раз

  нужна для сайд эффектов \*/

  useEffect(() => {

    dispatch(getRepos());

  }, []);

  return (

    <div>

      {repos.map((repo) => (

        <Repo repo={repo} />

      ))}

    </div>

  );

}

export default Main;

import axios from "axios";

import { setRepos } from "../reducers/reposReducer";

export const getRepos = (searchQuery /\* = "stars:%3E1" \*/) => {

  /\* Так как используем redux-thunk нужно создать ещё одну функцию \*/

  return async (dispatch) => {

    const response = await axios.get(

      `https://api.github.com/search/repositories?q=${searchQuery}&sort=stars`

    );

    dispatch(setRepos(response.data));

  };

};

import React from "react";

import "./repo.less";

function Repo(props) {

  const repo = props.repo;

  return (

    <div className="repo">

      <div className="repo-header">

        <div className="repo-header-name">{repo.name}</div>

        <div className="repo-header-stars">{repo.stargazers\_count}</div>

      </div>

      <div className="repo-last-commit">{repo.updated\_at}</div>

      <a href={repo.html\_url} className="repo-link">

        Ссылка на репозиторий

      </a>

    </div>

  );

}

export default Repo;

.repo {

  border: solid 2px lightgray;

  padding: 20px;

  display: flex;

  flex-direction: column;

}

&-header {

  display: flex;

  justify-content: space-between;

  margin-bottom: 20px;

  &-name {

    font-size: 1.2em;

    font-weight: 500;

  }

}

**6. React Redux - поиск репозиториев, анимация подгрузки**

/\* У каждого action есть type и хорошая практика выносить всё это в отдельную переменную \*/

/\* Загрузка данных с GitHub \*/

const SET\_REPOS = "SET\_REPOS";

/\* Loader \*/

const SET\_IS\_FETCHING = "SET\_IS\_FETCHING";

const defaultState = {

  /\* items полученные данные с github \*/

  items: [],

  /\* isFetching когда получим данные \*/

  isFetching: true,

};

export default function reposReducer(state = defaultState, action) {

  switch (action.type) {

    /\* Загрузка данных с GitHub \*/

    case SET\_REPOS:

      return {

        ...state,

        items: action.payload.items,

        isFetching: false,

      };

    case SET\_IS\_FETCHING:

      /\* Loader \*/

      return {

        ...state,

        isFetching: action.payload,

      };

    default:

      return state;

  }

}

/\* Также создадим Action creator это функция которая возращаем нам action

Это объект у которого есть поле type и какие либо данные

\*/

/\* Загрузка данных с GitHub \*/

export const setRepos = (repos) => ({ type: SET\_REPOS, payload: repos });

/\* Loader \*/

export const setIsFetching = (bool) => ({

  type: SET\_IS\_FETCHING,

  payload: bool,

});

import axios from "axios";

import { setIsFetching, setRepos } from "../reducers/reposReducer";

export const getRepos = (searchQuery /\* = "stars:%3E1" \*/) => {

  /\* Так как используем redux-thunk нужно создать ещё одну функцию \*/

  return async (dispatch) => {

    dispatch(setIsFetching(true));

    const response = await axios.get(

      `https://api.github.com/search/repositories?q=${searchQuery}&sort=stars`

    );

    dispatch(setRepos(response.data));

  };

};

import React, { useEffect, useState } from "react";

import "./main.less";

import { useDispatch, useSelector } from "react-redux";

import { getRepos } from "../actions/repos";

import Repo from "./repo/repo";

function Main() {

  const dispatch = useDispatch();

  /\* Получаю данные с GitHub \*/

  const repos = useSelector((state) => state.repos.items);

  /\* Loader \*/

  const isFetching = useSelector((state) => state.repos.isFetching);

  /\* Делаем поисковый инпут управляемым \*/

  const [searchValue, setSearchValue] = useState("");

  /\* Если [] зависимости пустой то вызовится один раз

  нужна для сайд эффектов \*/

  useEffect(() => {

    dispatch(getRepos());

  }, []);

  /\* При нажатии на кнопку будет получать репозиторий но с нашим поисковым запросом\*/

  function searchHandler() {

    dispatch(getRepos(searchValue));

  }

  return (

    <div>

      <div className="search">

        <input

          value={searchValue}

          onChange={(e) => setSearchValue(e.target.value)}

          type="text"

          placeholder="Input repo name"

          className="search-input"

        />

        <button onClick={() => searchHandler()} className="search-btn">

          Search

        </button>

      </div>

      {isFetching === false ? (

        repos.map((repo) => <Repo repo={repo} />)

      ) : (

        <div className="fetching"></div>

      )}

    </div>

  );

}

export default Main;

.search {

  display: flex;

  margin: 10px 0;

  &-input {

    padding: 5px;

    width: 100%;

  }

  &-btn {

    padding: 5px;

  }

}

.fetching {

  width: 100px;

  height: 100px;

  border-radius: 100%;

  margin: 30vh auto;

  border: 2px dashed red;

  /\* Чтобы анимация работала надо укзаать её имя и длительность \*/

  animation-name: search-fetching;

  animation-duration: 0.5s;

  /\* Чтобы анимация не заканчивалась \*/

  animation-iteration-count: infinite;

  /\* Чтобы анимация была туда обратно \*/

  animation-direction: alternate;

  /\* Чтобы скорость анимации во всех промежутках была одинакова \*/

  animation-timing-function: linear;

}

/\* Устанавливаем значение анимации с 0 до 360 за 1 секунду \*/

@keyframes search-fetching {

  0% {

    transform: rotate(0deg);

  }

  100% {

    transform: rotate(360deg) scale(1.3);

  }

}

import axios from "axios";

import { setIsFetching, setRepos } from "../reducers/reposReducer";

export const getRepos = (searchQuery /\* = "stars:%3E1" \*/) => {

  /\* Так как используем redux-thunk нужно создать ещё одну функцию \*/

  /\* Если инпут пустой то присвоим дефотное значение \*/

  if (searchQuery == "") {

    searchQuery = "stars:%3E1";

  }

  return async (dispatch) => {

    dispatch(setIsFetching(true));

    const response = await axios.get(

      `https://api.github.com/search/repositories?q=${searchQuery}&sort=stars`

    );

    dispatch(setRepos(response.data));

  };

};

**7. React Redux - Pagination\Пагинация - постраничный вывод репозиториев GitHub**

/\* У каждого action есть type и хорошая практика выносить всё это в отдельную переменную \*/

/\* Загрузка данных с GitHub \*/

const SET\_REPOS = "SET\_REPOS";

/\* Loader \*/

const SET\_IS\_FETCHING = "SET\_IS\_FETCHING";

/\* Текущая страница \*/

const SET\_CURRENT\_PAGE = "SET\_CURRENT\_PAGE";

const defaultState = {

  /\* items полученные данные с github \*/

  items: [],

  /\* isFetching когда получим данные \*/

  isFetching: true,

  /\* Номер текущей страницы \*/

  currentPage: 1,

  perPage: 10,

  /\* Все полученные от gitHub репозитории \*/

  totalCount: 0,

};

export default function reposReducer(state = defaultState, action) {

  switch (action.type) {

    /\* Загрузка данных с GitHub \*/

    case SET\_REPOS:

      return {

        ...state,

        items: action.payload.items,

        totalCount: action.payload.total\_count,

        isFetching: false,

      };

    case SET\_IS\_FETCHING:

      /\* Loader \*/

      return {

        ...state,

        isFetching: action.payload,

      };

    case SET\_CURRENT\_PAGE:

      /\* Текущая страница \*/

      return {

        ...state,

        currentPage: action.payload,

      };

    default:

      return state;

  }

}

/\* Также создадим Action creator это функция которая возращаем нам action

Это объект у которого есть поле type и какие либо данные

\*/

/\* Загрузка данных с GitHub \*/

export const setRepos = (repos) => ({ type: SET\_REPOS, payload: repos });

/\* Loader \*/

export const setIsFetching = (bool) => ({

  type: SET\_IS\_FETCHING,

  payload: bool,

});

/\* Текущая страница \*/

export const setCurrentPage = (page) => ({

  type: SET\_CURRENT\_PAGE,

  payload: page,

});

import React, { useEffect, useState } from "react";

import "./main.less";

import { useDispatch, useSelector } from "react-redux";

import { getRepos } from "../actions/repos";

import Repo from "./repo/repo";

function Main() {

  const dispatch = useDispatch();

  /\* Получаю данные с GitHub \*/

  const repos = useSelector((state) => state.repos.items);

  /\* Loader \*/

  const isFetching = useSelector((state) => state.repos.isFetching);

  /\* Текущая страница \*/

  const currentPage = useSelector((state) => state.repos.currentPage);

  /\*Всего страниц \*/

  const totalCount = useSelector((state) => state.repos.totalCount);

  /\* Текущая страница \*/

  const perPage = useSelector((state) => state.repos.perPage);

  /\* Страницы \*/

  const pages = [1, 2, 3, 4, 5];

  /\* Делаем поисковый инпут управляемым \*/

  const [searchValue, setSearchValue] = useState("");

  /\* Если [] зависимости пустой то вызовится один раз

  нужна для сайд эффектов \*/

  useEffect(() => {

    dispatch(getRepos());

  }, []);

  /\* При нажатии на кнопку будет получать репозиторий но с нашим поисковым запросом\*/

  function searchHandler() {

    dispatch(getRepos(searchValue));

  }

  return (

    <div>

      <div className="search">

        <input

          value={searchValue}

          onChange={(e) => setSearchValue(e.target.value)}

          type="text"

          placeholder="Input repo name"

          className="search-input"

        />

        <button onClick={() => searchHandler()} className="search-btn">

          Search

        </button>

      </div>

      {isFetching === false ? (

        repos.map((repo) => <Repo repo={repo} />)

      ) : (

        <div className="fetching"></div>

      )}

      {/\* Отображаем наши страницы \*/}

      <div className="pages">

        {pages.map((page, index) => (

          <span key={index} className="page">

            {page}

          </span>

        ))}

      </div>

    </div>

  );

}

export default Main;

import React, { useEffect, useState } from "react";

import "./main.less";

import { useDispatch, useSelector } from "react-redux";

import { getRepos } from "../actions/repos";

import Repo from "./repo/repo";

function Main() {

  const dispatch = useDispatch();

  /\* Получаю данные с GitHub \*/

  const repos = useSelector((state) => state.repos.items);

  /\* Loader \*/

  const isFetching = useSelector((state) => state.repos.isFetching);

  /\* Текущая страница \*/

  const currentPage = useSelector((state) => state.repos.currentPage);

  /\*Всего страниц \*/

  const totalCount = useSelector((state) => state.repos.totalCount);

  /\* Текущая страница \*/

  const perPage = useSelector((state) => state.repos.perPage);

  /\* Страницы \*/

  const pages = [1, 2, 3, 4, 5];

  /\* Делаем поисковый инпут управляемым \*/

  const [searchValue, setSearchValue] = useState("");

  /\* Если [] зависимости пустой то вызовится один раз

  нужна для сайд эффектов \*/

  /\* currentPage добавили для того что при изменение наша зависимсоть изменяется \*/

  useEffect(() => {

    dispatch(getRepos(searchValue));

  }, [currentPage]);

  /\* При нажатии на кнопку будет получать репозиторий но с нашим поисковым запросом\*/

  function searchHandler() {

    dispatch(getRepos(searchValue));

  }

  return (

    <div>

      <div className="search">

        <input

          value={searchValue}

          onChange={(e) => setSearchValue(e.target.value)}

          type="text"

          placeholder="Input repo name"

          className="search-input"

        />

        <button onClick={() => searchHandler()} className="search-btn">

          Search

        </button>

      </div>

      {isFetching === false ? (

        repos.map((repo) => <Repo repo={repo} />)

      ) : (

        <div className="fetching"></div>

      )}

      {/\* Отображаем наши страницы \*/}

      <div className="pages">

        {pages.map((page, index) => (

          <span

            key={index}

            /\* Если текущая страница то свой класс \*/

            className={currentPage == page ? "current-page" : "page"}

            /\* При нажатии изменяем номер страницы \*/

            onClick={() => dispatch(currentPage(page))}

          >

            {page}

          </span>

        ))}

      </div>

    </div>

  );

}

export default Main;

import axios from "axios";

import { setIsFetching, setRepos } from "../reducers/reposReducer";

export const getRepos = (searchQuery = "stars:%3E1", currentPage, perPage) => {

  /\* Так как используем redux-thunk нужно создать ещё одну функцию \*/

  /\* Если инпут пустой то присвоим дефотное значение \*/

  if (searchQuery == "") {

    searchQuery = "stars:%3E1";

  }

  return async (dispatch) => {

    dispatch(setIsFetching(true));

    const response = await axios.get(

      `https://api.github.com/search/repositories?q=${searchQuery}&sort=stars&per\_page=${perPage}&page=${currentPage}`

    );

    dispatch(setRepos(response.data));

  };

};

import React, { useEffect, useState } from "react";

import "./main.less";

import { useDispatch, useSelector } from "react-redux";

import { getRepos } from "../actions/repos";

import Repo from "./repo/repo";

import { setCurrentPage } from "../reducers/reposReducer";

function Main() {

  const dispatch = useDispatch();

  /\* Получаю данные с GitHub \*/

  const repos = useSelector((state) => state.repos.items);

  /\* Loader \*/

  const isFetching = useSelector((state) => state.repos.isFetching);

  /\* Текущая страница \*/

  const currentPage = useSelector((state) => state.repos.currentPage);

  /\*Всего страниц \*/

  const totalCount = useSelector((state) => state.repos.totalCount);

  /\* Колличество репозиториев на странице \*/

  const perPage = useSelector((state) => state.repos.perPage);

  /\* Страницы \*/

  const pages = [1, 2, 3, 4, 5];

  /\* Делаем поисковый инпут управляемым \*/

  const [searchValue, setSearchValue] = useState("");

  /\* Если [] зависимости пустой то вызовится один раз

  нужна для сайд эффектов \*/

  /\* currentPage добавили для того что при изменение наша зависимсоть изменяется \*/

  useEffect(() => {

    dispatch(getRepos(searchValue, currentPage, perPage));

  }, [currentPage]);

  /\* При нажатии на кнопку будет получать репозиторий но с нашим поисковым запросом\*/

  function searchHandler() {

    dispatch(setCurrentPage(1));

    dispatch(getRepos(searchValue, currentPage, perPage));

  }

  return (

    <div>

      <div className="search">

        <input

          value={searchValue}

          onChange={(e) => setSearchValue(e.target.value)}

          type="text"

          placeholder="Input repo name"

          className="search-input"

        />

        <button onClick={() => searchHandler()} className="search-btn">

          Search

        </button>

      </div>

      {isFetching === false ? (

        repos.map((repo) => <Repo repo={repo} />)

      ) : (

        <div className="fetching"></div>

      )}

      {/\* Отображаем наши страницы \*/}

      <div className="pages">

        {pages.map((page, index) => (

          <span

            key={index}

            /\* Если текущая страница то свой класс \*/

            className={currentPage == page ? "current-page" : "page"}

            /\* При нажатии изменяем номер страницы \*/

   onClick={() => dispatch(setCurrentPage(page))}

          >

            {page}

          </span>

        ))}

      </div>

    </div>

  );

}

export default Main;

создал utils а в нем pagesCreator

export function cratePages(pages, pagesCount, currentPage) {

  if (pagesCount > 10) {

    if (currentPage > 5) {

      for (let i = currentPage - 4; i <= currentPage + 5; i++) {

        pages.push(i);

        if (i == pagesCount) break;

      }

    } else {

      for (let i = 1; i <= 10; i++) {

        pages.push(i);

        if (i == pagesCount) break;

      }

    }

  } else {

    for (let i = 1; i <= pagesCount; i++) {

      pages.push(i);

    }

  }

}

import React, { useEffect, useState } from "react";

import "./main.less";

import { useDispatch, useSelector } from "react-redux";

import { getRepos } from "../actions/repos";

import Repo from "./repo/repo";

import { setCurrentPage } from "../reducers/reposReducer";

import { cratePages } from "../utils/pagesCreator";

function Main() {

  const dispatch = useDispatch();

  /\* Получаю данные с GitHub \*/

  const repos = useSelector((state) => state.repos.items);

  /\* Loader \*/

  const isFetching = useSelector((state) => state.repos.isFetching);

  /\* Текущая страница \*/

  const currentPage = useSelector((state) => state.repos.currentPage);

  /\*Всего страниц \*/

  const totalCount = useSelector((state) => state.repos.totalCount);

  /\* Колличество репозиториев на странице \*/

  const perPage = useSelector((state) => state.repos.perPage);

  /\* Общее колличество страниц \*/

  const pagesCount = Math.ceil(totalCount / perPage);

  /\* Страницы \*/

  const pages = [];

  /\* Вызываем функцию с логикой переключения страниц

  передадим массив страниц, общее колличество страниц и текущую страницу

  \*/

  cratePages(pages, pagesCount, currentPage);

  /\* Делаем поисковый инпут управляемым \*/

  const [searchValue, setSearchValue] = useState("");

  /\* Если [] зависимости пустой то вызовится один раз

  нужна для сайд эффектов \*/

  /\* currentPage добавили для того что при изменение наша зависимсоть изменяется \*/

  useEffect(() => {

    dispatch(getRepos(searchValue, currentPage, perPage));

  }, [currentPage]);

  /\* При нажатии на кнопку будет получать репозиторий но с нашим поисковым запросом\*/

  function searchHandler() {

    dispatch(setCurrentPage(1));

    dispatch(getRepos(searchValue, currentPage, perPage));

  }

  return (

    <div>

      <div className="search">

        <input

          value={searchValue}

          onChange={(e) => setSearchValue(e.target.value)}

          type="text"

          placeholder="Input repo name"

          className="search-input"

        />

        <button onClick={() => searchHandler()} className="search-btn">

          Search

        </button>

      </div>

      {isFetching === false ? (

        repos.map((repo) => <Repo repo={repo} />)

      ) : (

        <div className="fetching"></div>

      )}

      {/\* Отображаем наши страницы \*/}

      <div className="pages">

        {pages.map((page, index) => (

          <span

            key={index}

            /\* Если текущая страница то свой класс \*/

            className={currentPage == page ? "current-page" : "page"}

            /\* При нажатии изменяем номер страницы \*/

            onClick={() => dispatch(setCurrentPage(page))}

          >

            {page}

          </span>

        ))}

      </div>

    </div>

  );

}

export default Main;

**8. React Redux - Роутинг\Маршрутизация по страницам. React-router-dom, Router, Switch, Redirect**

Создаем папку card и в ней card.js

import React from "react";

function Card() {

  return <div>Card</div>;

}

export default Card;

import React from "react";

import "./app.less";

import { BrowserRouter, Route, Routes } from "react-router-dom";

import Main from "./main/Main";

import Card from "./card/card";

function App() {

  return (

    <BrowserRouter>

      <div className="container">

        <Routes>

          <Route path="/" element={<Main />} />

          <Route path="/card" element={<Card />} />

        </Routes>

      </div>

    </BrowserRouter>

  );

}

export default App;

/\* Импортируем из node.js библиотеку path которая позволит нам указать путь к папке \*/

const path = require("path");

const { CleanWebpackPlugin } = require("clean-webpack-plugin");

const HTMLWebpackPlugin = require("html-webpack-plugin");

module.exports = {

  /\* mode - указывает что находимсяв  разработке и выходные данные JS не надо сжимать

  entry - затем надо указать путь к файлу который будет входным с которого будет начинаться запуск приложения

  output - куда будет сам вебпак собирать файлы

  \*/

  mode: "development",

  entry: ["@babel/polyfill", "./src/index.jsx"],

  output: {

    /\* path - путь куда webpack будет делать сборку папка dist \*/

    path: path.resolve(\_\_dirname, "dist"),

    /\* Имя файла куда вебпак будет делать сборку всех файлов \*/

    /\* Будем использовать регулярные выражения чтобы не было проблем при хэшировании разработки чтобы он делал бандлы с уникальными названиями\*/

    filename: "[name].[hash].js",

  },

  /\* Настраиваем devServer \*/

  devServer: {

    port: 3000,

**historyApiFallback: true,**

  },

  resolve: {

    extensions: [".js", ".jsx"],

  },

  /\* Теперь подключаем плагины указываем путь к HTML \*/

  plugins: [

    new HTMLWebpackPlugin({ template: "./src/index.html" }),

    new CleanWebpackPlugin(),

  ],

  /\* Настраиваем ипорт наших стилей

  для этого задаем определенные правила \*/

  module: {

    rules: [

      {

        /\* Будет обрабатывать указанные ниже loader css и less файлы\*/

        test: /\.(css|less)$/,

        /\* Указываем массив loader который будет использоваться с лева на право учитываем \*/

        use: ["style-loader", "css-loader", "less-loader"],

      },

      {

        /\* Делаем обработку для файлов \*/

        test: /\.(jpg|jpeg|png|svg)$/,

        use: ["file-loader"],

      },

      {

        /\* Правила из Babel \*/

        test: /\.m?js$/,

        exclude: /node\_modules/,

        use: {

          loader: "babel-loader",

          options: {

            presets: ["@babel/preset-env"],

          },

        },

      },

      {

        /\* Скопировали добавили jsx и пресет для реакта\*/

        test: /\.m?jsx$/,

        exclude: /node\_modules/,

        use: {

          loader: "babel-loader",

          options: {

            presets: ["@babel/preset-env", "@babel/preset-react"],

          },

        },

      },

    ],

  },

};

import React from "react";

import { useNavigate } from "react-router-dom";

function Card() {

  /\* Навигация по страницам \*/

  const navigate = useNavigate();

  return (

    <div>

      {/\* При клике возвращаемся на предыдущую страницу \*/}

      <button onClick={() => navigate(-1)} className="back-btn">

        BACK

      </button>

      Card

    </div>

  );

}

export default Card;

import React from "react";

import "./repo.less";

import { NavLink } from "react-router-dom";

function Repo(props) {

  const repo = props.repo;

  return (

    <div className="repo">

      <div className="repo-header">

        <div className="repo-header-name">

          <NavLink to={"/card"}>{repo.name}</NavLink>

        </div>

        <div className="repo-header-stars">{repo.stargazers\_count}</div>

      </div>

      <div className="repo-last-commit">{repo.updated\_at}</div>

      <a href={repo.html\_url} className="repo-link">

        Ссылка на репозиторий

      </a>

    </div>

  );

}

export default Repo;

**9. React Redux - страница репозитория.**

/\* Импортируем из node.js библиотеку path которая позволит нам указать путь к папке \*/

const path = require("path");

const { CleanWebpackPlugin } = require("clean-webpack-plugin");

const HTMLWebpackPlugin = require("html-webpack-plugin");

module.exports = {

  /\* mode - указывает что находимсяв  разработке и выходные данные JS не надо сжимать

  entry - затем надо указать путь к файлу который будет входным с которого будет начинаться запуск приложения

  output - куда будет сам вебпак собирать файлы

  \*/

  mode: "development",

  entry: ["@babel/polyfill", "./src/index.jsx"],

  output: {

    /\* path - путь куда webpack будет делать сборку папка dist \*/

    path: path.resolve(\_\_dirname, "dist"),

    /\* Имя файла куда вебпак будет делать сборку всех файлов \*/

    /\* Будем использовать регулярные выражения чтобы не было проблем при хэшировании разработки чтобы он делал бандлы с уникальными названиями\*/

    filename: "[name].[hash].js",

**/\* Чтобы не было проблем с кэшированием при переходах на разные страницы \*/**

**publicPath: "/",**

  },

  /\* Настраиваем devServer \*/

  devServer: {

    port: 3000,

    historyApiFallback: true,

  },

  resolve: {

    extensions: [".js", ".jsx"],

  },

  /\* Теперь подключаем плагины указываем путь к HTML \*/

  plugins: [

    new HTMLWebpackPlugin({ template: "./src/index.html" }),

    new CleanWebpackPlugin(),

  ],

  /\* Настраиваем ипорт наших стилей

  для этого задаем определенные правила \*/

  module: {

    rules: [

      {

        /\* Будет обрабатывать указанные ниже loader css и less файлы\*/

        test: /\.(css|less)$/,

        /\* Указываем массив loader который будет использоваться с лева на право учитываем \*/

        use: ["style-loader", "css-loader", "less-loader"],

      },

      {

        /\* Делаем обработку для файлов \*/

        test: /\.(jpg|jpeg|png|svg)$/,

        use: ["file-loader"],

      },

      {

        /\* Правила из Babel \*/

        test: /\.m?js$/,

        exclude: /node\_modules/,

        use: {

          loader: "babel-loader",

          options: {

            presets: ["@babel/preset-env"],

          },

        },

      },

      {

        /\* Скопировали добавили jsx и пресет для реакта\*/

        test: /\.m?jsx$/,

        exclude: /node\_modules/,

        use: {

          loader: "babel-loader",

          options: {

            presets: ["@babel/preset-env", "@babel/preset-react"],

          },

        },

      },

    ],

  },

};

import React from "react";

import { useNavigate, useParams } from "react-router-dom";

function Card() {

  /\* Навигация по страницам \*/

  const navigate = useNavigate();

  /\* Позволяет получать параметры из url \*/

  const { username, reponame } = useParams();

  return (

    <div>

      {/\* При клике возвращаемся на предыдущую страницу \*/}

      <button onClick={() => navigate(-1)} className="back-btn">

        BACK

      </button>

      Card

    </div>

  );

}

export default Card;

import React from "react";

import "./app.less";

import { BrowserRouter, Route, Routes, Navigate } from "react-router-dom";

import Main from "./main/Main";

import Card from "./card/Card";

function App() {

  return (

    <BrowserRouter>

      <div className="container">

        <Routes>

          <Route exact path="/" element={<Main />} />

          {/\* Получим :username/:reponame через хук useParams \*/}

          <Route path="/card/:username/:reponame" element={<Card />} />

          {/\* Редирект если вбиваем не точный адерсс \*/}

          <Route path="\*" element={<Navigate to="/" />} />

        </Routes>

      </div>

    </BrowserRouter>

  );

}

export default App;

import React from "react";

import "./repo.less";

import { NavLink } from "react-router-dom";

function Repo(props) {

  const repo = props.repo;

  return (

    <div className="repo">

      <div className="repo-header">

        <div className="repo-header-name">

          <NavLink to={`/card/${repo.owner.login}`}>{repo.name}</NavLink>

        </div>

        <div className="repo-header-stars">{repo.stargazers\_count}</div>

      </div>

      <div className="repo-last-commit">{repo.updated\_at}</div>

      <a href={repo.html\_url} className="repo-link">

        Ссылка на репозиторий

      </a>

    </div>

  );

}

export default Repo;

import axios from "axios";

import { setIsFetching, setRepos } from "../reducers/reposReducer";

export const getRepos = (searchQuery = "stars:%3E1", currentPage, perPage) => {

  /\* Так как используем redux-thunk нужно создать ещё одну функцию \*/

  /\* Если инпут пустой то присвоим дефотное значение \*/

  if (searchQuery == "") {

    searchQuery = "stars:%3E1";

  }

  return async (dispatch) => {

    dispatch(setIsFetching(true));

    const response = await axios.get(

      `https://api.github.com/search/repositories?q=${searchQuery}&sort=stars&per\_page=${perPage}&page=${currentPage}`

    );

    dispatch(setRepos(response.data));

  };

};

**/\* Берем с API отдельно взятый репозиторий и информацию о нём \*/**

**export const getCurrentRepo = async (username, repoName, setRepo) => {**

**const response = await axios.get(**

**`https://api.github.com/repos/${username}/${repoName}`**

**);**

**setRepo(response.data);**

**};**

import React, { useEffect, useState } from "react";

import { useNavigate, useParams } from "react-router-dom";

import { getCurrentRepo } from "../actions/repos";

function Card() {

  /\* Навигация по страницам \*/

  const navigate = useNavigate();

  /\* Позволяет получать параметры из url \*/

  const { username, reponame } = useParams();

  console.log(username, reponame);

  /\* Сам репоизторий \*/

  const [repo, setRepo] = useState({});

  /\* Вызовем функцию получающую репозиторий  и передадим в неё все необходимые параметры\*/

  useEffect(() => {

    getCurrentRepo(username, reponame, setRepo);

  }, []);

  return (

    <div>

      {/\* При клике возвращаемся на предыдущую страницу \*/}

      <button onClick={() => navigate(-1)} className="back-btn">

        BACK

      </button>

      {/\* Отобразим информацию о пользователе репозитория \*/}

      <div className="card">

        <img src="" alt="" />

        <div className="name">{repo.name}</div>

        <div className="stars">{repo.stargazers\_count}</div>

      </div>

    </div>

  );

}

export default Card;

.card {

  display: flex;

  margin-top: 10px;

}

.name {

  font-size: 1.5em;

  font-weight: 700;

  margin-left: 10px;

}

.stars {

  margin-left: auto;

}

import React, { useEffect, useState } from "react";

import { useNavigate, useParams } from "react-router-dom";

import { getCurrentRepo } from "../actions/repos";

import "./card.less";

function Card() {

  /\* Навигация по страницам \*/

  const navigate = useNavigate();

  /\* Позволяет получать параметры из url \*/

  const { username, reponame } = useParams();

  console.log(username, reponame);

  /\* Сам репоизторий \*/

  const [repo, setRepo] = useState({ owner: {} });

**/\* Контребьютерс \*/**

**const [contributors, setContributors] = useState();**

  /\* Вызовем функцию получающую репозиторий  и передадим в неё все необходимые параметры\*/

  useEffect(() => {

    getCurrentRepo(username, reponame, setRepo);

  }, []);

  return (

    <div>

      {/\* При клике возвращаемся на предыдущую страницу \*/}

      <button onClick={() => navigate(-1)} className="back-btn">

        BACK

      </button>

      {/\* Отобразим информацию о пользователе репозитория \*/}

      <div className="card">

        <img src={repo.owner.avatar\_url} alt="" />

        <div className="name">{repo.name}</div>

        <div className="stars">{repo.stargazers\_count}</div>

      </div>

    </div>

  );

}

export default Card;

import axios from "axios";

import { setIsFetching, setRepos } from "../reducers/reposReducer";

export const getRepos = (searchQuery = "stars:%3E1", currentPage, perPage) => {

  /\* Так как используем redux-thunk нужно создать ещё одну функцию \*/

  /\* Если инпут пустой то присвоим дефотное значение \*/

  if (searchQuery == "") {

    searchQuery = "stars:%3E1";

  }

  return async (dispatch) => {

    dispatch(setIsFetching(true));

    const response = await axios.get(

      `https://api.github.com/search/repositories?q=${searchQuery}&sort=stars&per\_page=${perPage}&page=${currentPage}`

    );

    dispatch(setRepos(response.data));

  };

};

/\* Берем с API отдельно взятый репозиторий и информацию о нём \*/

export const getCurrentRepo = async (username, reponame, setRepo) => {

  const response = await axios.get(

    `https://api.github.com/repos/${username}/${reponame}`

  );

  setRepo(response.data);

};

**/\* Берем с API contributors \*/**

**export const getCotributors = async (username, reponame, setContributors) => {**

**const response = await axios.get(**

**`https://api.github.com/repos/${username}/${reponame}/contributors?page=1&per\_page=10`**

**);**

**setContributors(response.data);**

**};**

import React, { useEffect, useState } from "react";

import { useNavigate, useParams } from "react-router-dom";

import { getCotributors, getCurrentRepo } from "../actions/repos";

import "./card.less";

function Card() {

  /\* Навигация по страницам \*/

  const navigate = useNavigate();

  /\* Позволяет получать параметры из url \*/

  const { username, reponame } = useParams();

  console.log(username, reponame);

  /\* Сам репоизторий \*/

  const [repo, setRepo] = useState({ owner: {} });

  /\* Контребьютерс \*/

  const [contributors, setContributors] = useState();

  /\* Вызовем функцию получающую репозиторий  и передадим в неё все необходимые параметры\*/

  useEffect(() => {

    getCurrentRepo(username, reponame, setRepo);

**getCotributors(username, reponame, setContributors);**

  }, []);

  return (

    <div>

      {/\* При клике возвращаемся на предыдущую страницу \*/}

      <button onClick={() => navigate(-1)} className="back-btn">

        BACK

      </button>

      {/\* Отобразим информацию о пользователе репозитория \*/}

      <div className="card">

        <img src={repo.owner.avatar\_url} alt="" />

        <div className="name">{repo.name}</div>

        <div className="stars">{repo.stargazers\_count}</div>

      </div>

**{/\* Показываем Контребьютерс \*/}**

**{contributors &&**

**contributors.map((c, index) => (**

**<div key={index}>**

**{index + 1}. {c.login}**

**</div>**

**))}**

    </div>

  );

}

export default Card;

**10. React Redux - обработка ошибок**

/\* У каждого action есть type и хорошая практика выносить всё это в отдельную переменную \*/

/\* Загрузка данных с GitHub \*/

const SET\_REPOS = "SET\_REPOS";

/\* Loader \*/

const SET\_IS\_FETCHING = "SET\_IS\_FETCHING";

/\* Текущая страница \*/

const SET\_CURRENT\_PAGE = "SET\_CURRENT\_PAGE";

**/\* Есть ошибка или нет \*/**

**const SET\_FETCH\_ERROR = "SET\_FETCH\_ERROR";**

const defaultState = {

  /\* items полученные данные с github \*/

  items: [],

  /\* isFetching когда получим данные \*/

  isFetching: true,

  /\* Номер текущей страницы \*/

  currentPage: 1,

  /\* Колличество репозиториев на странице \*/

  perPage: 10,

  /\* Все полученные от gitHub репозитории \*/

  totalCount: 0,

**/\* Найдена ошибка или нет \*/**

**isFetchError: false,**

};

export default function reposReducer(state = defaultState, action) {

  switch (action.type) {

    /\* Загрузка данных с GitHub \*/

    case SET\_REPOS:

      return {

        ...state,

        items: action.payload.items,

        totalCount: action.payload.total\_count,

        isFetching: false,

      };

    case SET\_IS\_FETCHING:

      /\* Loader \*/

      return {

        ...state,

        isFetching: action.payload,

      };

    case SET\_CURRENT\_PAGE:

      /\* Текущая страница \*/

      return {

        ...state,

        currentPage: action.payload,

      };

**case SET\_FETCH\_ERROR:**

**/\* Есть ошибка или нет \*/**

**return {**

**...state,**

**isFetchError: action.payload,**

**};**

    default:

      return state;

  }

}

/\* Также создадим Action creator это функция которая возращаем нам action

Это объект у которого есть поле type и какие либо данные

\*/

/\* Загрузка данных с GitHub \*/

export const setRepos = (repos) => ({ type: SET\_REPOS, payload: repos });

/\* Loader \*/

export const setIsFetching = (bool) => ({

  type: SET\_IS\_FETCHING,

  payload: bool,

});

/\* Текущая страница \*/

export const setCurrentPage = (page) => ({

  type: SET\_CURRENT\_PAGE,

  payload: page,

});

**/\* Есть ошибка или нет \*/**

**export const setFetchError = (bool) => ({**

**type: SET\_FETCH\_ERROR,**

**payload: bool,**

**});**

import React, { useEffect, useState } from "react";

import "./main.less";

import { useDispatch, useSelector } from "react-redux";

import { getRepos } from "../actions/repos";

import Repo from "./repo/repo";

import { setCurrentPage } from "../reducers/reposReducer";

import { cratePages } from "../utils/pagesCreator";

function Main() {

  const dispatch = useDispatch();

  /\* Получаю данные с GitHub \*/

  const repos = useSelector((state) => state.repos.items);

  /\* Loader \*/

  const isFetching = useSelector((state) => state.repos.isFetching);

  /\* Текущая страница \*/

  const currentPage = useSelector((state) => state.repos.currentPage);

  /\*Всего страниц \*/

  const totalCount = useSelector((state) => state.repos.totalCount);

  /\* Колличество репозиториев на странице \*/

  const perPage = useSelector((state) => state.repos.perPage);

**/\* Есть ошибка или нет \*/**

**const isFetchError = useSelector((state) => state.repos.isFetchError);**

  /\* Общее колличество страниц \*/

  const pagesCount = Math.ceil(totalCount / perPage);

  /\* Страницы \*/

  const pages = [];

  /\* Вызываем функцию с логикой переключения страниц

  передадим массив страниц, общее колличество страниц и текущую страницу

  \*/

  cratePages(pages, pagesCount, currentPage);

  /\* Делаем поисковый инпут управляемым \*/

  const [searchValue, setSearchValue] = useState("");

  /\* Если [] зависимости пустой то вызовится один раз

  нужна для сайд эффектов \*/

  /\* currentPage добавили для того что при изменение наша зависимсоть изменяется \*/

  useEffect(() => {

    dispatch(getRepos(searchValue, currentPage, perPage));

  }, [currentPage]);

  /\* При нажатии на кнопку будет получать репозиторий но с нашим поисковым запросом\*/

  function searchHandler() {

    dispatch(setCurrentPage(1));

    dispatch(getRepos(searchValue, currentPage, perPage));

  }

  return (

    <div>

      <div className="search">

        <input

          value={searchValue}

          onChange={(e) => setSearchValue(e.target.value)}

          type="text"

          placeholder="Input repo name"

          className="search-input"

        />

        <button onClick={() => searchHandler()} className="search-btn">

          Search

        </button>

      </div>

      {isFetching === false ? (

        repos.map((repo) => <Repo repo={repo} />)

      ) : (

        <div className="fetching"></div>

      )}

      {/\* Отображаем наши страницы \*/}

      <div className="pages">

        {pages.map((page, index) => (

          <span

            key={index}

            /\* Если текущая страница то свой класс \*/

            className={currentPage == page ? "current-page" : "page"}

            /\* При нажатии изменяем номер страницы \*/

            onClick={() => dispatch(setCurrentPage(page))}

          >

            {page}

          </span>

        ))}

      </div>

    </div>

  );

}

export default Main;

Рассмотрим два типа обработки ошибок редирект на страницу с информацией об ошибке

import React from "react";

import { useNavigate } from "react-router-dom";

function Error(props) {

  /\* Навигация по страницам \*/

  const navigate = useNavigate();

  return (

    <div style={{ textAlign: "center" }}>

      <button onClick={() => navigate(-1)}>GO TO MAIN PAGE</button>Error

    </div>

  );

}

export default Error;

import React from "react";

import "./app.less";

import { BrowserRouter, Route, Routes, Navigate } from "react-router-dom";

import Main from "./main/Main";

import Card from "./card/Card";

import Error from "./main/Error";

function App() {

  return (

    <BrowserRouter>

      <div className="container">

        <Routes>

          <Route exact path="/" element={<Main />} />

          {/\* Получим :username/:reponame через хук useParams \*/}

          <Route path="/card/:username/:reponame" element={<Card />} />

          {/\* При ошибке \*/}

          <Route path="/error" element={<Error />} />

          {/\* Редирект если вбиваем не точный адерсс \*/}

          <Route path="\*" element={<Navigate to="/" />} />

        </Routes>

      </div>

    </BrowserRouter>

  );

}

export default App;

import React, { useEffect, useState } from "react";

import "./main.less";

import { useDispatch, useSelector } from "react-redux";

import { getRepos } from "../actions/repos";

import Repo from "./repo/repo";

import { setCurrentPage } from "../reducers/reposReducer";

import { cratePages } from "../utils/pagesCreator";

function Main() {

  const dispatch = useDispatch();

  /\* Получаю данные с GitHub \*/

  const repos = useSelector((state) => state.repos.items);

  /\* Loader \*/

  const isFetching = useSelector((state) => state.repos.isFetching);

  /\* Текущая страница \*/

  const currentPage = useSelector((state) => state.repos.currentPage);

  /\*Всего страниц \*/

  const totalCount = useSelector((state) => state.repos.totalCount);

  /\* Колличество репозиториев на странице \*/

  const perPage = useSelector((state) => state.repos.perPage);

  /\* Есть ошибка или нет \*/

  const isFetchError = useSelector((state) => state.repos.isFetchError);

  /\* Общее колличество страниц \*/

  const pagesCount = Math.ceil(totalCount / perPage);

  /\* Страницы \*/

  const pages = [];

  /\* Вызываем функцию с логикой переключения страниц

  передадим массив страниц, общее колличество страниц и текущую страницу

  \*/

  cratePages(pages, pagesCount, currentPage);

  /\* Делаем поисковый инпут управляемым \*/

  const [searchValue, setSearchValue] = useState("");

  /\* Если [] зависимости пустой то вызовится один раз

  нужна для сайд эффектов \*/

  /\* currentPage добавили для того что при изменение наша зависимсоть изменяется \*/

  useEffect(() => {

    dispatch(getRepos(searchValue, currentPage, perPage));

  }, [currentPage]);

  /\* При нажатии на кнопку будет получать репозиторий но с нашим поисковым запросом\*/

  function searchHandler() {

    dispatch(setCurrentPage(1));

    dispatch(getRepos(searchValue, currentPage, perPage));

  }

**/\* Если условие с ошибкой true То редирект на страницу с ошибкой \*/**

**if (isFetchError) {**

**return <Redirect to="/error" />;**

**}**

  return (

    <div>

      <div className="search">

        <input

          value={searchValue}

          onChange={(e) => setSearchValue(e.target.value)}

          type="text"

          placeholder="Input repo name"

          className="search-input"

        />

        <button onClick={() => searchHandler()} className="search-btn">

          Search

        </button>

      </div>

      {isFetching === false ? (

        repos.map((repo) => <Repo repo={repo} />)

      ) : (

        <div className="fetching"></div>

      )}

      {/\* Отображаем наши страницы \*/}

      <div className="pages">

        {pages.map((page, index) => (

          <span

            key={index}

            /\* Если текущая страница то свой класс \*/

            className={currentPage == page ? "current-page" : "page"}

            /\* При нажатии изменяем номер страницы \*/

            onClick={() => dispatch(setCurrentPage(page))}

          >

            {page}

          </span>

        ))}

      </div>

    </div>

  );

}

export default Main;

import axios from "axios";

import {

  setFetchError,

  setIsFetching,

  setRepos,

} from "../reducers/reposReducer";

export const getRepos = (searchQuery = "stars:%3E1", currentPage, perPage) => {

  /\* Так как используем redux-thunk нужно создать ещё одну функцию \*/

  /\* Если инпут пустой то присвоим дефотное значение \*/

  if (searchQuery == "") {

    searchQuery = "stars:%3E1";

  }

  return async (dispatch) => {

    /\* Чтобы обрабатывать ошибки необходимо поместить код в блок try catch \*/

    try {

      dispatch(setIsFetching(true));

      const response = await axios.get(

        `https://api.github.com/search/repositories?q=${searchQuery}&sort=stars&per\_page=${perPage}&page=${currentPage}`

      );

      dispatch(setRepos(response.data));

    } catch (e) {

      dispatch(setFetchError(true));

      dispatch(setIsFetching(false));

    }

  };

};

/\* Берем с API отдельно взятый репозиторий и информацию о нём \*/

export const getCurrentRepo = async (username, reponame, setRepo) => {

  const response = await axios.get(

    `https://api.github.com/repos/${username}/${reponame}`

  );

  setRepo(response.data);

};

/\* Берем с API contributors \*/

export const getCotributors = async (username, reponame, setContributors) => {

  const response = await axios.get(

    `https://api.github.com/repos/${username}/${reponame}/contributors?page=1&per\_page=10`

  );

  setContributors(response.data);

};

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

**<!-- Подключаем стили bootstrap -->**

**<link**

**href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/css/bootstrap.min.css"**

**rel="stylesheet"**

**integrity="sha384-KK94CHFLLe+nY2dmCWGMq91rCGa5gtU4mk92HdvYe+M/SXH301p5ILy+dN9+nJOZ"**

**crossorigin="anonymous"**

**/>**

    <title>Document</title>

  </head>

  <body>

    <div id="root"></div>

  </body>

</html>

import React, { useEffect, useState } from "react";

import "./main.less";

import { useDispatch, useSelector } from "react-redux";

import { getRepos } from "../actions/repos";

import Repo from "./repo/repo";

import { setCurrentPage } from "../reducers/reposReducer";

import { cratePages } from "../utils/pagesCreator";

function Main() {

  const dispatch = useDispatch();

  /\* Получаю данные с GitHub \*/

  const repos = useSelector((state) => state.repos.items);

  /\* Loader \*/

  const isFetching = useSelector((state) => state.repos.isFetching);

  /\* Текущая страница \*/

  const currentPage = useSelector((state) => state.repos.currentPage);

  /\*Всего страниц \*/

  const totalCount = useSelector((state) => state.repos.totalCount);

  /\* Колличество репозиториев на странице \*/

  const perPage = useSelector((state) => state.repos.perPage);

  /\* Есть ошибка или нет \*/

  const isFetchError = useSelector((state) => state.repos.isFetchError);

  /\* Общее колличество страниц \*/

  const pagesCount = Math.ceil(totalCount / perPage);

  /\* Страницы \*/

  const pages = [];

  /\* Вызываем функцию с логикой переключения страниц

  передадим массив страниц, общее колличество страниц и текущую страницу

  \*/

  cratePages(pages, pagesCount, currentPage);

  /\* Делаем поисковый инпут управляемым \*/

  const [searchValue, setSearchValue] = useState("");

  /\* Если [] зависимости пустой то вызовится один раз

  нужна для сайд эффектов \*/

  /\* currentPage добавили для того что при изменение наша зависимсоть изменяется \*/

  useEffect(() => {

    dispatch(getRepos(searchValue, currentPage, perPage));

  }, [currentPage]);

  /\* При нажатии на кнопку будет получать репозиторий но с нашим поисковым запросом\*/

  function searchHandler() {

    dispatch(setCurrentPage(1));

    dispatch(getRepos(searchValue, currentPage, perPage));

  }

  /\* Если условие с ошибкой true То редирект на страницу с ошибкой \*/

  /\*   if (isFetchError) {

    return <Redirect to="/error" />;

  } \*/

  return (

    <div>

**{/\* Если isFetchError то отображаем alert с ошибкой из bootstrap \*/}**

**{isFetchError && (**

**<div class="alert alert-danger" role="alert">**

**Произишла ошибка! Пожалуйста обновите страницу!**

**</div>**

**)}**

      <div className="search">

        <input

          value={searchValue}

          onChange={(e) => setSearchValue(e.target.value)}

          type="text"

          placeholder="Input repo name"

          className="search-input"

        />

        <button onClick={() => searchHandler()} className="search-btn">

          Search

        </button>

      </div>

      {isFetching === false ? (

        repos.map((repo) => <Repo repo={repo} />)

      ) : (

        <div className="fetching"></div>

      )}

      {/\* Отображаем наши страницы \*/}

      <div className="pages">

        {pages.map((page, index) => (

          <span

            key={index}

            /\* Если текущая страница то свой класс \*/

            className={currentPage == page ? "current-page" : "page"}

            /\* При нажатии изменяем номер страницы \*/

            onClick={() => dispatch(setCurrentPage(page))}

          >

            {page}

          </span>

        ))}

      </div>

    </div>

  );

}

export default Main;

import axios from "axios";

import {

  setFetchError,

  setIsFetching,

  setRepos,

} from "../reducers/reposReducer";

export const getRepos = (searchQuery = "stars:%3E1", currentPage, perPage) => {

  /\* Так как используем redux-thunk нужно создать ещё одну функцию \*/

  /\* Если инпут пустой то присвоим дефотное значение \*/

  if (searchQuery == "") {

    searchQuery = "stars:%3E1";

  }

  return async (dispatch) => {

    /\* Чтобы обрабатывать ошибки необходимо поместить код в блок try catch \*/

    try {

      dispatch(setIsFetching(true));

      const response = await axios.get(

        `https://api.github.com/search/repositories?q=${searchQuery}&sort=stars&per\_page=${perPage}&page=${currentPage}`

      );

      dispatch(setRepos(response.data));

    } catch (e) {

      dispatch(setFetchError(true));

      dispatch(setIsFetching(false));

**/\* Убираем сообщение об ошибке через две секунды \*/**

**setTimeout(() => {**

**dispatch(setFetchError(false));**

**}, 2000);**

    }

  };

};

/\* Берем с API отдельно взятый репозиторий и информацию о нём \*/

export const getCurrentRepo = async (username, reponame, setRepo) => {

  const response = await axios.get(

    `https://api.github.com/repos/${username}/${reponame}`

  );

  setRepo(response.data);

};

/\* Берем с API contributors \*/

export const getCotributors = async (username, reponame, setContributors) => {

  const response = await axios.get(

    `https://api.github.com/repos/${username}/${reponame}/contributors?page=1&per\_page=10`

  );

  setContributors(response.data);

};