Пререквизиты

Для начала работы на вашем компьютере должны быть:

1. Установлен Visual Studio Code. Скачать можно тут https://code.visualstudio.com/
2. Установлен Node.js. Скачиваем тут https://nodejs.org/en/
3. Установлен npm или yarn. Yarn скачиваем тут https://yarnpkg.com/en/docs/install
4. Расширение Redux Dev Tools для Chrome
5. Хорошее настроение, время, налитый горячий чай или кофе.

Опционально: плагин для VS CODE "Markdown all in one" https://marketplace.visualstudio.com/items?itemName=yzhang.markdown-all-in-one

Начало работы

1. Открываем проект в Visual Studio Code.
2. Устанавливаем зависимости npm install (yarn)
3. Запускаем проект npm run start (yarn run start)
4. Открываем наш проект на порту 8080 – http://localhost:8080
5. Видим проект со статьями

Используем React Hooks

React Hooks — это функции, с помощью которых вы можете «подцепиться» к состоянию и методам жизненного цикла React из функциональных компонентов.

В компоненте `News.jsx` вместо класса используем функцию:

```
const News = () => {
    if (isLoading) { // isLoading ?
        return <div className="loader" />;
    }

    return news.map(( // news ?
        item,
```

```
        index
    ) => <ItemComponent key={item.id} index={index} item={item} />);
};
```

useState

Используем `useState` чтобы наделить наш функциональный компонент внутренним состоянием. Единственный аргумент useState — это начальное состояние.

```
import React, { useState } from "react"; // импортируем useState

...

const News = () => {
  const [news, setNews] = useState([]);
  const [isLoading, setIsLoading] = useState([]);

  ...
}
```

Вызов useState возвращает две вещи: текущее значение состояния и функцию для его обновления.

useEffect

Теперь необходимо сделать так, чтобы при отображении компонента у нас подгружались данные. В классовом компоненте мы делали это, используя life cycle метод класса `componentDidMount`:

```
// вызывается сразу после монтирования (то есть, вставки компонента в DOM)
componentDidMount() {
    this.setState({ isLoading: true })
   fetch(`${API_URL}/search_by_date`)
        .then(response => response.json())
        .then(response => response.hits)
        .then(json => this.setState({ news: json, isLoading: false }));
}
```

А сейчас, давайте рассмотрим, как мы можем сделать то же самое с использованием хука `useEffect`.

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

const News = () => {
  const [news, setNews] = useState([]);
  const [isLoading, setIsLoading] = useState(false);

  useEffect(() => {
      setIsLoading(true);
      fetch(`${API_URL}/search_by_date`)
```

```
            .then(response => response.json())
            .then(response => response.hits)
            .then(json => {
                setNews(json);
                setIsLoading(false);
            });
    }, []);

    return {...}
```

React запомнит функцию (то есть «эффект»), которую вы передали и вызовет её после того, как внесёт все изменения в DOM. `useEffect` имеет второй необязательный параметр - массив зависимостей. Если мы хотим, чтобы эффект запустился один раз, то необходимо передать пустой массив.

Custom Hook

Было бы здорово вынести всю логику в отдельный хук, чтобы наш компонент выглядел так:

```
  import React from "react";
  import useNews from './hooks/useNews'

  const News = () => {
    const {isLoading, news} = useNews();

    return {...}
```

Для этого делаем свой кастомный хук `useNews` и вынесем всю логику получения данных в неё. Для этого создадим новый файл `useNews.js` в папке `./hooks`:

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

import { API_URL } from "../constants";

const useNews = () => {
    const [news, setNews] = useState([]);
    const [isLoading, setIsLoading] = useState(false);

    useEffect(() => {
        setIsLoading(true);
        fetch(`${API_URL}/search_by_date`)
            .then(response => response.json())
            .then(response => response.hits)
            .then(json => {
                setNews(json);
                setIsLoading(false);
            });
    }, []);

    return { news, isLoading };
```

```
};

export default useNews;
```

Попробуйте подключить его в файле `News.jsx`, вместо `useState` и `useEffect`:

```
const { isLoading, news } = useNews();
```

Добавляем панель фильтров

В папке `/FiltersBar` добавим строку поиска, фильтр по категориям и сортировку:

Файл `SearchBar.jsx`:

```
import React, { useState } from "react";
import { DebounceInput } from "react-debounce-input";

function SearchBar() {
    const [search, setSearch] = useState("");

    return (
        <DebounceInput
            className="search-bar"
            minLength={2}
            debounceTimeout={500}
            onChange={event => setSearch(event.target.value)}
        />
    );
}

export default SearchBar;
```

Для строки поиска мы будем использовать `DebounceInput`, что поможет нам уменьшить количество запросов на сервер.

Файл `CategoryFilter.jsx`:

```
import React, { useState } from "react";
import Select from "react-select";

import { CATEGORIES_OPTIONS } from "../constants";

function CategoryFilter() {
    const [category, setCategory] = useState(null);

    return (
        <Select
```

```
            className="category-filter"
            isMulti
            value={category}
            onChange={setCategory}
            options={CATEGORIES_OPTIONS}
            placeholder="Select category"
        />
    );
}

export default CategoryFilter;
```

Файл `Sorting.jsx`:

```
import React, { useState } from "react";
import Select from "react-select";

import { SORT_OPTIONS } from "../constants";

function Sorting() {
    const [sort, setSort] = useState(SORT_OPTIONS[0]);

    return (
        <Select
            className="sort-filter"
            value={sort}
            onChange={setSort}
            options={SORT_OPTIONS}
            placeholder="Select sorting"
        />
    );
}

export default Sorting;
```

В файле `News.jsx` добавим панель фильтров:

```
...
import FiltersBar from './FiltersBar/FiltersBar';

const News = () => {
    const { isLoading, news } = useNews();

    return (
        <>
            <FiltersBar />
            {!isLoading ? news.map((item, index) => (
                <ItemComponent key={item.id} index={index} item={item} />
            )) : <div className="loader" />}
```

```
        </>
    );
};
```

Для того, чтобы использовать фильтры, нам необходимо вынести данные в общее хранилище, для этого будем использовать Redux.

Подключаем Redux

Инициализация

Чтобы использовать Redux необходимо обернуть всё приложение в компонент `<Provider>`, чтобы Redux-стор был доступен в дереве компонентов.

В файле `index.js` добавим код:

```
import React from "react";
import ReactDOM from "react-dom";
import { createStore } from "redux";
import { Provider } from "react-redux";

import rootReducer from "./reducers/rootReducer";

const root = document.createElement("container");
document.body.appendChild(root);

const store = createStore(rootReducer, window?.__REDUX_DEVTOOLS_EXTENSION__);

const render = () => {
    ReactDOM.render(
        <Provider store={store}>
            <App />
        </Provider>,
        root
    );
};

render();
```

Actions

Далее переместим хранение новостей и фильтров в Redux-store. Начнём с создания actions. Для этого в папке `actions` создадим файл `actions.js`:

```
export const SET_SEARCH_FILTER = 'SET_SEARCH_FILTER';
export const SET_CATEGORY_FILTER = 'SET_CATEGORY_FILTER';
export const SET_SORTING = 'SET_SORTING';
```

```
export const FETCH_NEWS_SUCCESS = 'FETCH_NEWS_SUCCESS';

// action creators:

export const setSearchFilter = search => ({
    type: SET_SEARCH_FILTER,
    payload: { search },
});

export const setCategoryFilter = category => ({
    type: SET_CATEGORY_FILTER,
    payload: { category },
});

export const setSorting = sort => ({
    type: SET_SORTING,
    payload: { sort },
});

export const setNews = news => ({
    type: FETCH_NEWS_SUCCESS,
    payload: { news },
});
```

Reducers

В папке `reducers` редьюсеры для фильтров и новостей. Вместо конструкции `switch` используем хелпер `createReducer`:

`newsReducer.js`

```
import { createReducer } from '@reduxjs/toolkit';

import { FETCH_NEWS_SUCCESS } from '../actions/actions';

const initialState = {
    items: [],
};

const newsReducer = createReducer(initialState, {
    [FETCH_NEWS_SUCCESS]: (state, action) => ({
        ...state,
        items: action.payload.news,
    }),
});

export default newsReducer;
```

filtersReducer.js

```
import { createReducer } from '@reduxjs/toolkit';

import { SET_SEARCH_FILTER, SET_CATEGORY_FILTER, SET_SORTING } from
'../actions/actions';
import { SORT_OPTIONS } from '../constants';

const initialState = {
    search: '',
    sort: SORT_OPTIONS[0],
    category: null,
};

const filtersReducer = createReducer(initialState, {
    [SET_SEARCH_FILTER]: (state, action) => ({
        ...state,
        search: action.payload.search,
    }),
    [SET_CATEGORY_FILTER]: (state, action) => ({
        ...state,
        category: action.payload.category,
    }),
    [SET_SORTING]: (state, action) => ({
        ...state,
        sort: action.payload.sort,
    }),
});

export default filtersReducer;
```

И объединим их в файле rootReducer.js, используя combineReducers, которое позволяет разбивать стор на отдельные модули:

```
import { combineReducers } from 'redux';

import filtersReducer from './filtersReducer';
import newsReducer from './newsReducer';

const rootReducer = combineReducers({
    filters: filtersReducer,
    news: newsReducer,
});

export default rootReducer;
```

Redux Hooks

Для работы с Redux-store нам понадобятся хуки `useSelector` - для получения данных с redux store и `useDispatch` для получения ссылки на `dispatch` функции.

```
import { useState, useEffect } from 'react';
import { useSelector, useDispatch } from 'react-redux';

import { API_URL } from '../constants';
import { setNews } from '../actions/actions';

const useNews = () => {
    const [isLoading, setIsLoading] = useState(false);
    const dispatch = useDispatch();
    const news = useSelector(state => state.news.items);

    useEffect(() => {
        setIsLoading(true);
        fetch(`${API_URL}/search_by_date`)
            .then(response => response.json())
            .then(response => response.hits)
            .then((json) => {
                dispatch(setNews(json));
                setIsLoading(false);
            });
    }, [dispatch]);

    return { news, isLoading };
};

export default useNews;
```

Создадим файл `useFilters.js` для описания хуков для фильтров:

```
import { useSelector, useDispatch } from 'react-redux';
import { useCallback } from 'react';

import { setSearchFilter, setSorting, setCategoryFilter } from
'../actions/actions';

export const useSearch = () => {
    const dispatch = useDispatch();
    const search = useSelector(state => state.filters.search);

    return [search, useCallback(value => dispatch(setSearchFilter(value)),
[dispatch])];
};

export const useCategory = () => {
    const dispatch = useDispatch();
    const category = useSelector(state => state.filters.category);
```

```
    return [category, useCallback(value => dispatch(setCategoryFilter(value)),
[dispatch])];
};

export const useSort = () => {
    const dispatch = useDispatch();
    const sort = useSelector(state => state.filters.sort);

    return [sort, useCallback(value => dispatch(setSorting(value)), [dispatch])];
};
```

`useCallback` вернёт мемоизированную версию колбэка, который изменяется только, если изменяются значения одной из зависимостей. Это полезно при передаче колбэков оптимизированным дочерним компонентам, которые полагаются на равенство ссылок для предотвращения ненужных рендеров

В соответствующих компонентах фильтров делаем замену на наши хуки:

```
    // CategoryFilter.jsx
    const [category, setCategory] = useCategory();

    // SearchBar.jsx
    const [search, setSearch] = useSearch();

    // Sorting.jsx
    const [sort, setSort] = useSort();
```

Проверяем работу в Redux Dev Tools.

Добавляем фильтры в `useNews`;

```
import { useState, useEffect } from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { stringify } from 'query-string';

import { API_URL } from '../constants';
import { setNews } from '../actions/actions';

import { useSearch, useCategory, useSort } from './useFilters';

const useNews = () => {
    const [isLoading, setIsLoading] = useState(false);
    const dispatch = useDispatch();
    const news = useSelector(state => state.news.items);
    // используем фильтры
    const [search] = useSearch();
    const [category] = useCategory();
    const [sort] = useSort();
```

```
    useEffect(() => {
        setIsLoading(true);
        fetch(`${API_URL}/${sort.value}?${stringify({
            query: search, tags: category ? category.map(x => x.value).join(',') :
''',
        })}`)
            .then(response => response.json())
            .then(response => response.hits)
            .then((json) => {
                dispatch(setNews(json));
                setIsLoading(false);
            });
    }, [category, dispatch, search, sort]);

    return { news, isLoading };
};

export default useNews;
```

Обратите внимание на массив зависимостей в `useEffect`, при изменении этих параметров новости будут перезагружаться заново.