Столыпинский вестник

Научная статья

Original article

УДК 004

ИССЛЕДОВАНИЕ СОВРЕМЕННОГО СТЕКА REACT РАЗРАБОТКИ В 2022 ГОДУ

THE STUDY OF MODERN GLASS REACT DEVELOPMENTS IN 2022

Берьянов Максим Сергеевич, Магистрант, 2 курс, Факультет ПИиКТ, Университет ИТМО Россия, г. Санкт-Петербург

Салахов Илья Равильевич, Магистрант, 2 курс, Факультет ПИиКТ, Университет ИТМО, Россия, г. Санкт-Петербург

Иванов Михаил Дмитриевич, Магистрант, 2 курс, Факультет ПИиКТ, Университет ИТМО Россия, г. Санкт-Петербург

Buryanov Maxim Sergeevich, Master's student, 2nd year, Faculty of Pmik, ITMO University Russia, St. Petersburg

Ilya Ravilevich Salakhov, Master's student, 2nd year, Faculty of Pmikt, ITMO University, Russia, St. Petersburg

Mikhail Dmitrievich Ivanov, Master's student, 2nd year, Faculty of Pmik, ITMO University Russia, St. Petersburg

Аннотация. Разработка на React.JS, несомненно, актуальна в современном мире. С течением времени появляются новые инструменты, происходят ключевые

изменения в подходах к проектированию систем, разработке веб-концепций. В данной статье предлагается исследовать современный стек продвинутой React-разработки 2022 года, внутри которого присутствуют такие важные технологии и инструменты, как TypeScript, Tailwind CSS, Redux Toolkit и Redux Toolkit Query. Текущее исследование представляет актуальность как для продвинутых, так и для начинающих веб-исследователей.

Abstract. React.JS development is undoubtedly relevant in the modern world. Eventually, new tools appear, key changes occur in system design approaches and web concepts development. This article proposes to explore the modern stack of advanced React development in 2022, within which there are such important technologies and tools as TypeScript, Tailwind CSS, Redux Toolkit and Redux Toolkit Query. The current research is of relevance to both advanced and novice web researchers.

Ключевые слова: веб-разработка, React.JS, Tailwind CSS, Redux Toolkit, TypeScript.

Keywords: web development, React.JS, Tailwind CSS, Redux Toolkit, TypeScript.

Стоит начать с того, чем является библиотека React.JS [1]. Это инструмент, который позволяет удобным образом создавать пользовательские интерфейсы. Можно упомянуть основные принципы, которых придерживается React.JS:

– Декларативность (довольно просто создавать интерфейсы, части которых будут автоматически обновляться самим React при изменении данных).
– Концепция компонентов (возможно создавать инкапсулированные компоненты с собственным состоянием, а затем объединять их в сложные пользовательские интерфейсы).

На момент написания данной статьи актуальная версия React.JS – 18.2.0. Более того, в текущее время на второй план при разработке уходит использование

чистого JavaScript [2] – вместо этого, как и в этой статье, будет использован язык TypeScript [3], который является оберткой над JavaScript и предоставляет возможность явной типизации, что действительно помогает при создании и поддержке больших проектов.

В качестве среды разработки предлагается использовать продукт IntelliJ IDEA Ultimate [4] версии 2022.2.3.

В первую очередь, необходимо проверить, установлен ли на компьютере инструмент Node.JS [5] с помощью команд *node -v* и *npm -v* в терминале. В случае отображения установленной версии, можно продолжать дальнейшую настройку проекта, иначе Node.JS нужно скачать с официального сайта (Node Package Manager установится автоматически).

Для того, чтобы сгенерировать проект на React, необходимо использовать утилиту create-react-app [6] с помощью команды: *npx create-react-app --template typescript . .* Данная технология облегчает нам задачу по настройке множества аспектов разработки, к примеру, таких как сборщик проекта, его окружение и другие. Иными словами, запустив данную команду мы получаем уже готовый проект «из коробки». Структуру сгенерированного проекта можно немного почистить, удалив ненужное для наших целей, конечный вариант представлен на рисунке 1. Данная структура характерна, можно сказать, для каждого frontend-приложения. В файле *package.json* в поле *dependencies* содержатся нужные нам зависимости для работы, в том числе, *react, react-dom* и их TypeScript-определения *@types/react, @types/react-dom*.

Более того, есть базовые скрипты, которые позволяют нам работать с данным проектом.

В качестве CSS в данной статье будет использована технология Tailwind CSS [7]. Устанавливается она последовательным запуском следующих команд: *npm install -D tailwindcss postcss autoprefixer,* затем *npx tailwindcss init -p,* а также

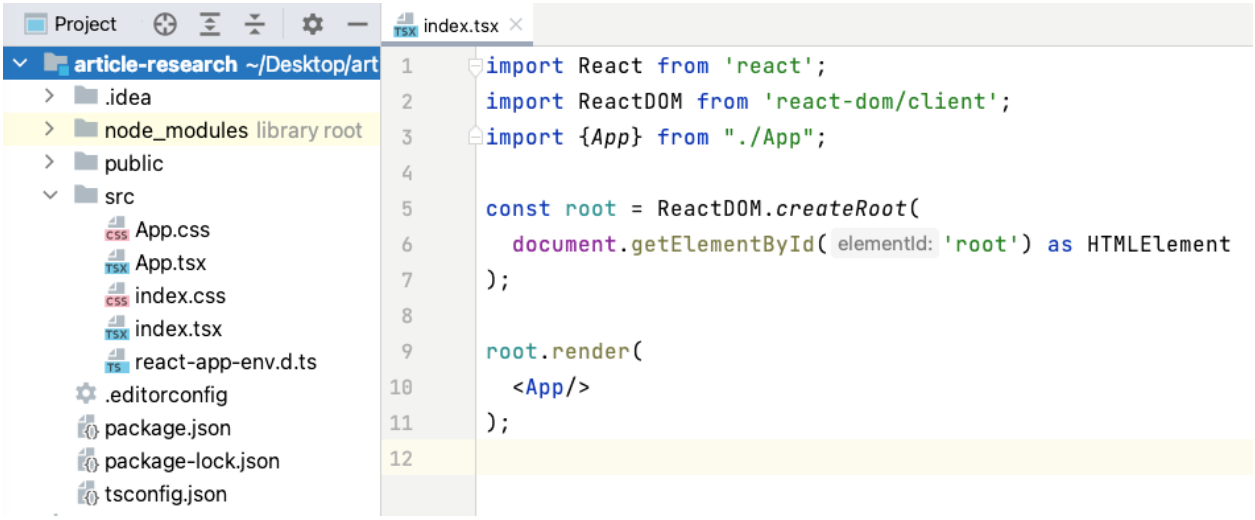

Рисунок 1 – Структура проекта

определением обрабатываемых элементов в файле *tailwind.config.js* через внутреннее поле *content: ["./src/**/*.{js,jsx,ts,tsx}"]*.

Более того, необходимо добавить директивы *@tailwind base*, *@tailwind components*, *@tailwind utilities* в главный CSS-файл, к примеру, *index.css*. Что касается самого Tailwind CSS, это CSS-библиотека, которая упрощает стилизацию HTML, тем же путем, как это делает Bootstrap [8], – добавляя огромное количество разнообразных классов. Но, в отличие от Bootstrap, который предлагает уже готовые к употреблению компоненты, такие как кнопки, alert'ы и navbar'ы, классы Tailwind CSS нацелены на конкретное свойство – в Tailwind CSS нет заранее написанной кнопки, её нужно написать самому. Конечно, считается, что у данной технологии существуют свои минусы, которые мы попытаемся опровергнуть:

– Как бы много ни было классов, мы всё равно ограничены данным нам набором, следовательно, загоняем оформление сайта в придуманные «рамки», что приводит к тому, что все сайты, написанные на Tailwind CSS похожи друг на друга.
– У нас множество лишних стилей, которые мы, скорее всего, не используем.

– Tailwind CSS нарушает концепцию не помещать множество inline стилей в HTML файл, хоть и косвенно.

В чем основная сила Bootstrap – он хорошо подходит для Proof Of Concept. Быстро нарисовать красивый и внятный интерфейс, когда нужно показать функционал – это Bootstrap. Нам не столь важен внешний вид, главное, чтобы выглядело приятно. Но при желании все компоненты можно изменить до неузнаваемости, переписав классы под свои нужды и добавив классы из чистого CSS. Касаемо Tailwind CSS, как было сказано ранее, готовых компонентов там нет, однако также есть ограничения, которые с легкостью решаются настройкой файла *tailwind.config.js* (к примеру, есть *margin* со значением 2.5rem, есть 3rem, но нет 2.75rem).

Проблема с переизбытком лишних классов также решается с помощью *tailwind.config.js*. Если раньше нужно было добавлять инструмент PurgeCSS [9] или другую библиотеку для удаления лишних классов, теперь Tailwind CSS сделает это за нас. Всё, что нам нужно сделать, – указать папку, где хранятся наши файлы с разметкой, что мы уже, собственно, и сделали при настройке. Tailwind CSS проанализирует файлы, найдет все использованные классы, оставит только их, а остальные удалит.

Более того, касаемо последнего минуса, при правильной реализации, описывать стилизацию вместе с разметкой оказывается невероятно удобно – видеть HTML-элемент и сразу понимать, как он стилизован, а также писать верстку, не перескакивая из файла в файл, нужно лишь поменять отношение к данной проблеме.

Установим также дополнительные зависимости для работы с помощью команды *npm install @reduxjs/toolkit react-redux react-router-dom*. Начнем с описания того, зачем нужна последняя зависимость React Router [10] – она отвечает

за перемещение между страницами (компонентами) в проекте (браузере). В React Router существует 3 категории компонентов:

– Роутеры (<BrowserRouter> или <HashRouter>).
– Маршруты (<Route>).
– Навигация (<Link>).

Любая маршрутизация начинается с роутера. Для веб-проектов react-router-dom предоставляет <BrowserRouter> и <HashRouter>. Основное отличие между ними состоит в способе хранения URL и взаимодействия с сервером.

<BrowserRouter> использует обычные URL. В этом случае URL выглядят как обычно, но требуется определенная настройка сервера. В частности, сервер должен обслуживать все страницы, используемые на клиенте. Create React App поддерживает это из коробки в режиме разработки и содержит инструкции для правильной настройки сервера.

<HashRouter> хранит текущую локацию в хэш-части URL (после специфического символа фрагмента "#"), поэтому URL выглядит примерно так: http://example.com/#/your/page. Поскольку хэш не отправляется серверу, его специальная настройка не требуется.

```
const root = ReactDOM.createRoot(
  document.getElementById( elementId: 'root') as HTMLElement
);

root.render(
  <BrowserRouter>
    <App/>
  </BrowserRouter>
);
```

Рисунок 2 – Роутер

Для использования роутера необходимо обернуть в него компонент верхнего уровня (рисунок 2). Стоит заметить, что на текущий момент актуальная версия React Router – 6. Отныне вместо компонента Switch появился компонент Routes. Но это не просто переименование — Routes более функционален. Основное отличие в том, что Routes не требует жесткого порядка роутов внутри. Switch обходил роуты в строгом порядке сверху вниз и при первом совпадении пути рендерил заданный компонент. Поэтому важно было определить порядок: например, выносить вниз

наиболее общий роут. Также, раньше в компоненте Route был выбор: либо указать компонент для рендера, либо передать render-prop функцию. В новой версии роутера оба этих свойства заменены на один — element. В него можно передать любой JSX-элемент.

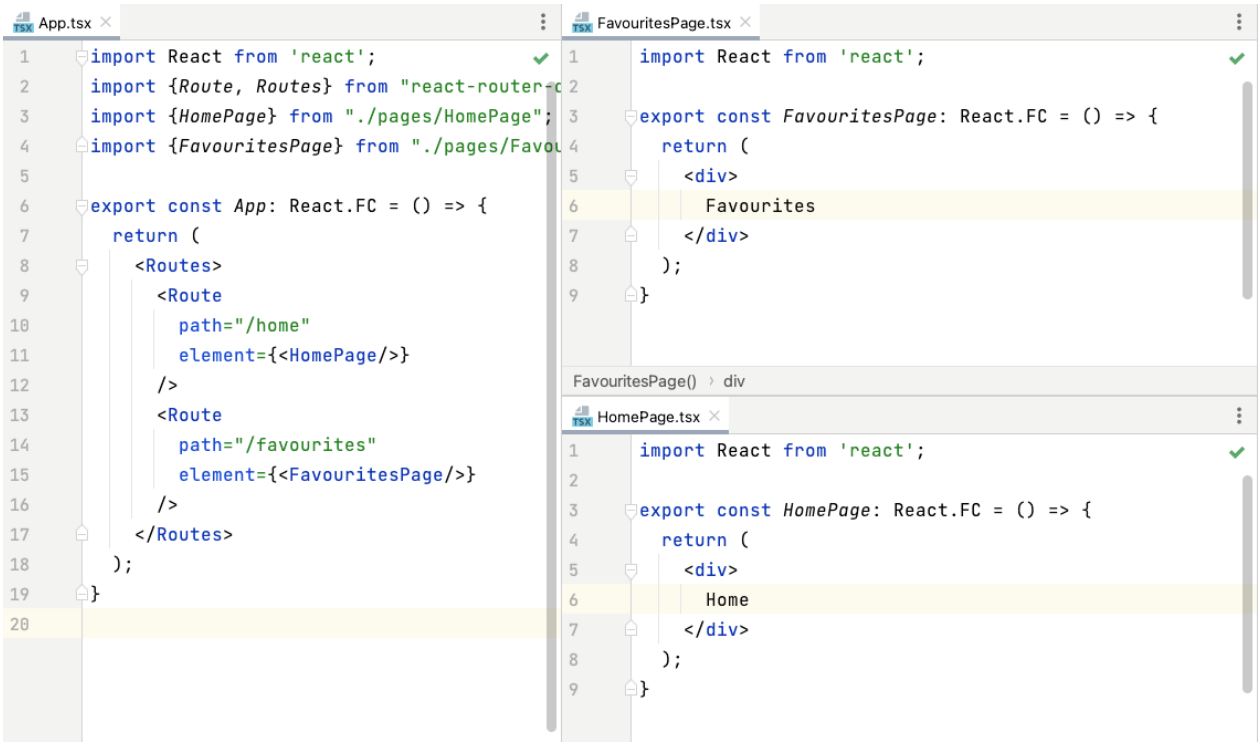

Рисунок 3 – Описание страниц и путей к ним

Создадим два контейнера, которые на самом деле будут разными страницами – по пути */home*, который указывается в свойстве *path* будет открываться компонент домашней страницы (пока что там обычный текст), а по пути */favourites* – компонент страницы избранного (рисунок 3). Стоит заметить, что далее работать мы будем с открытым API GitHub.

Можем увидеть промежуточный результат на рисунке 4. Перемещаясь между

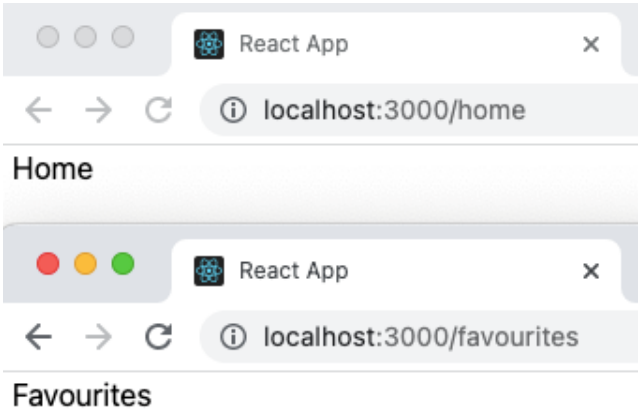

Рисунок 4 – Изменение пути

страницами, видно, как меняется содержимое. Однако, на данный момент такое поведение можно достичь, если менять ссылку в браузере собственноручно.

Рисунок 5 – Компонент навигационного бара

Конечно же, это неудобно. Поэтому мы должны создать некоторый навигационный бар, который позволит по нажатию на определенные компоненты-кнопки перемещать нас между страницами. Здесь как раз можно будет оценить преимущества использования обозначенного выше инструмента Tailwind CSS. В компоненте Navigation (рисунок 5) в атрибуте *className* через пробел указаны именно классы Tailwind CSS. Удобство их в том, что вместо того, чтобы создавать свой отдельный CSS-класс и определять в нем свойства *display: flex* и другие, гораздо приятнее использовать уже созданный за нас класс. Для того, чтобы понимать, за что отвечает каждый из них, необходимо использовать документацию Tailwind CSS (рисунок 6).

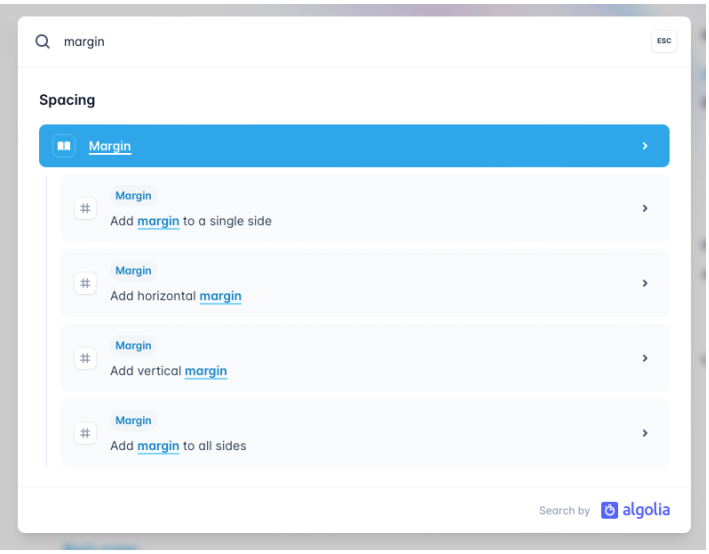

Рисунок 6 – CSS-свойство в Tailwind CSS

В поиске требуется ввести любое свойство, которое мы хотим переиспользовать, к примеру *margin*, а на выходе получить описание того, как данное свойство представлено в данной технологии (рисунок 7).

Рисунок 7 – Документация по свойству

Касаемо ссылок, используется компонент под названием Link. Он принимает в себя свойство *to* и позволяет по клику на дочерний компонент, обернутый собой, переходить по переданному адресу, изменяя в том числе содержимое адресной строки браузера. В нашем случае это две кнопки, которые ведут на домашнюю

страницу и страницу с избранным (рисунок 8). Также на данном рисунке можно увидеть собственный класс, который с помощью правила *@apply* применяет внутренние стили Tailwind CSS и тем самым показывает пример того, что правильно оформленный с помощью данной библиотеки код не повторяется.

Navigation.tsx

```tsx
<div className="flex">
  <Link to="/home" className="link-button">
    <div className="flex items-center">
      <IoHome className="mr-2"/>
      Home
    </div>
  </Link>
  <Link to="/favourites"
        className="link-button last-of-type:mr-[-0.75rem]">
    <div className="flex items-center">
      <MdOutlineFavorite className="mr-2"/>
      Favourites
    </div>
  </Link>
</div>
</nav>
```

NavigationStyles.css

```css
.link-button {
  @apply px-2 py-1 hover:bg-white
         hover:text-black hover:rounded-md;
}
```

Рисунок 8 – Навигационные кнопки

Конечный результат панели навигации можно увидеть на рисунке 9.

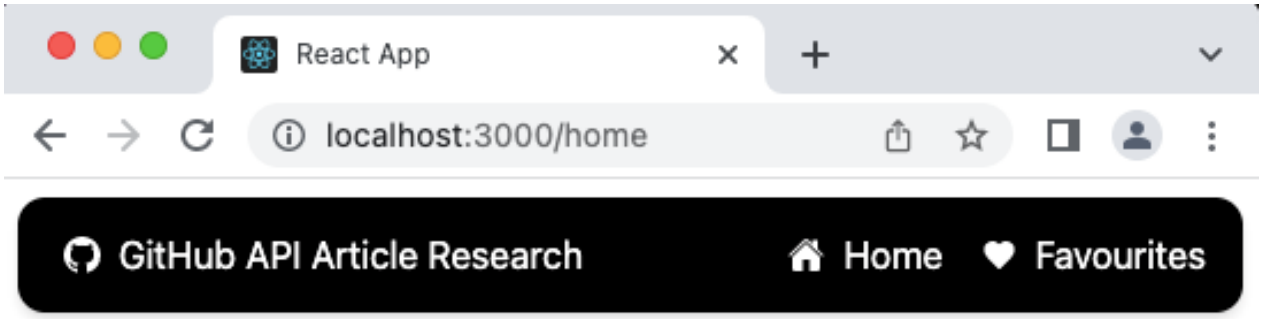

Рисунок 9 – Навигационная панель

Следующим шагом нашего исследования будет небольшая работа с данными, а именно их загрузка через GitHub API. Необходимо обучить наше приложение работе с Redux [11]. Что касается истории, механизм локального хранилища компонента, который поставляется вместе с базовой библиотекой (React) неудобен тем, что такое хранилище изолировано. К примеру, если мы хотим, чтобы разные независимые компоненты реагировали на какое-либо событие, нам придётся либо передавать локальное состояние в виде props'ов дочерним компонентам, либо поднимать его вверх до ближайшего родительского компонента. В обоих случаях делать это неудобно. Код становится более трудночитаемым, а компоненты зависимыми от их вложенности. Технология Redux снимает эту проблему так как состояние доступно всем компонентам без особых трудностей. Данная библиотека является самой популярной реализацией FLUX-архитектуры и, несмотря на ряд очевидных преимуществ, имеет весьма существенные недостатки, такие как:

– Сложность и "многословность" рекомендованных паттернов для написания и организации кода, что влечет за собой большое количество избыточности.
– Отсутствие встроенных средств управления асинхронным поведением и побочными эффектами, что приводит к необходимости выбора подходящего инструмента из множества дополнений, написанных сторонними разработчиками.

Для устранения этих недостатков разработчики Redux представили библиотеку Redux Toolkit [12]. Этот инструмент представляет собой набор практических решений и методов, предназначенных для упрощения разработки приложений с использованием Redux. Разработчики данной библиотеки преследовали цель упростить типичные случаи использования Redux. Данный инструмент не является универсальным решением в каждом из возможных случаев

использования Redux, но позволяет упростить тот код, который требуется написать разработчику.

Наиболее значимыми функциями, предоставляемыми библиотекой Redux Toolkit являются:

- configureStore – функция, предназначенная упростить процесс создания и настройки хранилища.
- createReducer – функция, помогающая лаконично и понятно описать, и создать reducer.
- createAction – функция, создающая действие для заданной строки типа действия.
- createSlice – функция, объединяющая в себе функционал createAction и createReducer.

Для настройки хранилища воспользуемся функцией configureStore. Данный инструмент позволяет автоматически комбинировать reducer’ы, добавить middleware’ы Redux (по умолчанию включает redux-thunk), а также использовать расширение Redux DevTools. В качестве входных параметров функция configureStore принимает объект со следующими важными свойствами:

- reducer – набор пользовательских reducer’ов.
- middleware – опциональный параметр, задающий массив middleware’ов, предназначенных для подключения к хранилищу.
- preloadedState – опциональный параметр, задающий начальное состояние хранилища.

Для получения наиболее популярного списка middleware’ов можно воспользоваться специальной функцией getDefaultMiddleware, также входящей в состав Redux Toolkit. Данная функция возвращает массив с включенными по умолчанию в библиотеку Redux Toolkit middleware’ами. Перечень этих middleware’ов отличается в зависимости от того, в каком режиме выполняется код.

Функция createReducer упрощает создание функций reducer’a, определяя их как таблицы поиска функций для обработки каждого типа действия. Она также позволяет существенно упростить логику неизменяемого обновления, написав код в изменяемом стиле внутри reducer’ов.

Изменяемый стиль обработки событий доступен благодаря использованию библиотеки Immer [13]. Функция обработчик может либо видоизменять переданное состояние (state) для изменения свойств, либо возвращать новый state, как при работе в неизменяемом стиле, но, благодаря Immer, реальное изменение объекта не осуществляется. Первый вариант куда проще для работы и восприятия, особенно при изменении объекта с глубокой вложенностью.

В качестве входных параметров функция createSlice принимает объект со следующими полями:

– name – пространство имен создаваемых действий (${name}/${action.type}).
– initialState – начальное состояние reducer’a.
– reducers – объект с обработчиками. Каждый обработчик принимает функцию с аргументами state и action, action содержит в себе данные в свойстве payload и имя события в свойстве name. Кроме того, имеется возможность предварительного изменения данных, полученных из события, перед их попаданием в reducer. Для этого вместо функции необходимо передать объект с полями reducer и prepare, где reducer — это функция-обработчик действия, a prepare — функция-обработчик полезной нагрузки, возвращающая обновленный payload.

Результатом работы функции является объект, называемый "срез", со следующими полями:

– name – имя среза.
– reducer – непосредственно, reducer.

– actions – набор действий.

Несмотря на то, что библиотека Redux Toolkit не вносит ничего нового в управление хранилищем, она предоставляет ряд гораздо более удобных средств для написания кода, чем были до этого. Данные средства позволяют не только сделать процесс разработки более удобным, понятным и быстрым, но и более эффективным, за счет наличия в библиотеке ряда хорошо зарекомендовавших себя ранее инструментов.

Также, существует инструмент Redux Toolkit Query [14], который помогает организовать работу с REST API. RTK Query включает следующие API:

– createApi – основная функциональность RTK Query, позволяет определить набор endpoint’ов, описать, как извлекать присылаемые данные, включая настройку того, как преобразовывать эти данные.
– fetchBaseQuery – небольшая обертка над fetch, предназначенная для упрощения запросов.
– setupListeners – утилита, используемая для включения поведения refetchOnMount и refetchOnReconnect.

Используем вышеизложенные инструменты в нашем исследовании и приведем определение API вызова поиска пользователей (рисунок 10). В качестве базового URL используется *https://api.github.com*, который помещается в поле *baseUrl* функции *createApi*. В качестве *endpoint* указываем собственно определяемое имя *searchUsers*, где в поле *url* помещаем относительный путь *search/users* с параметром адресной строки *q*, как этого требует данное API GitHub.

```
export const githubApi = createApi({
  reducerPath: 'github/api',
  baseQuery: fetchBaseQuery( {baseUrl, prepareHeaders, fetchFn, paramsSerializer, ...baseFetchOptions}: {
    baseUrl: 'https://api.github.com/'
  }),
  endpoints: build => ({
    searchUsers: build.query<IUser[], string>( definition: {
      query: (id: string) => ({
        url: 'search/users',
        params: {
          q: id
        }
      }),
      transformResponse: (response: ServerResponse<IUser>) => response.items
    })
  })
});

export const {
  useSearchUsersQuery
} = githubApi;
```

Рисунок 10 – Определение API поиска пользователей

Верстку страницы с поиском мы опустим ради уменьшения объема статьи, приведем лишь конечный результат (рисунок 11). Также, если открыть вкладку *Сеть* в Chrome DevTools, то можно увидеть, как происходит запрос (рисунок 12).

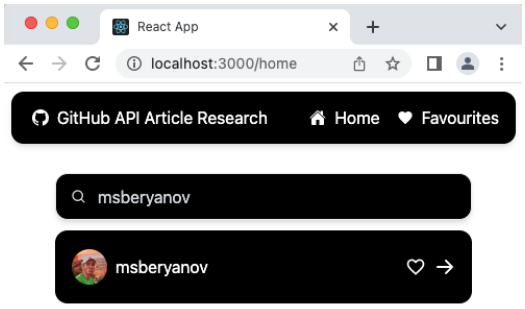

Рисунок 11 – Поиск пользователей

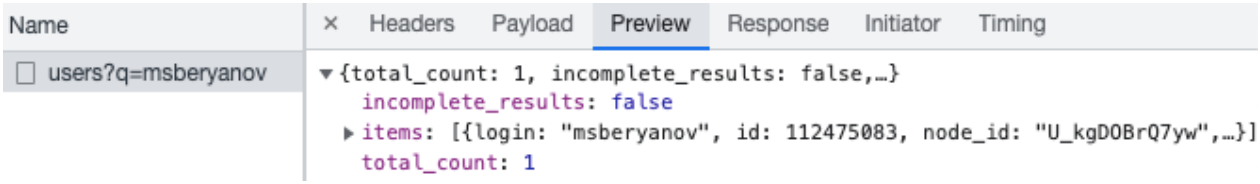

Рисунок 12 – Запрос в Chrome DevTools

Видим, что к данному запросу автоматически добавляется упомянутый параметр строки после вопросительного знака. Более того, стоит сказать, что для создания TypeScript-интерфейсов из JSON есть удобный сервис *json2ts* [15].

Создадим срез для хранения списка избранных пользователей (рисунок 13). Заметим, что вместе с определением начального состояния, описываются предполагаемые для работы reducer'ы – их actions затем можно экспортировать вызовом одноименного поля сущности *githubSice*. Данные действия можно связать с другими, если такие имеются, через функцию *bindActionCreators*, чтобы затем, используя созданный hook *useActions* импортировать оттуда методы добавления и удаления объектов из избранного, которые можно привязать к событию *onClick* по соответствующему значку на карточке пользователя.

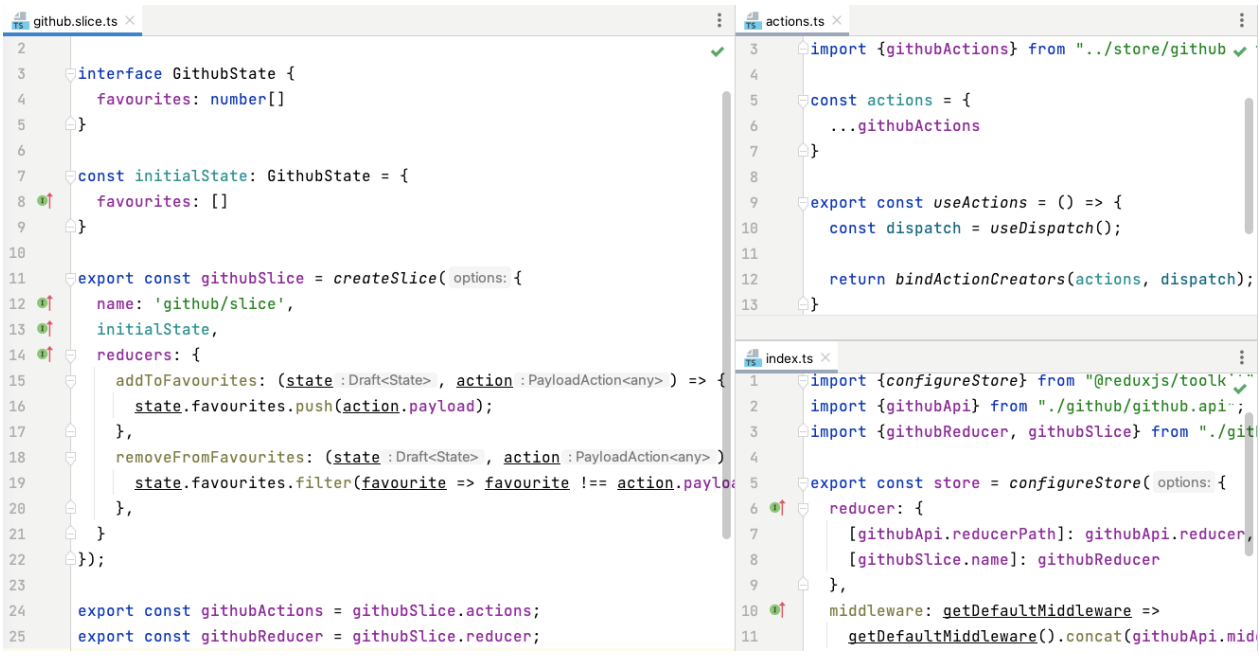

Рисунок 13 – Срез для управления состоянием списка избранного

Доступ к списку избранного определяется же вызовом собственного hook'а *useAppSelector*, который расширяет hook *useSelector* из библиотеки *react-redux* путем определения возвращаемого типа через интерфейс *TypedUseSelectorHook*. Таким образом, можно получать сущности любого состояния по названию среза (рисунок 14). При этом все срезы привязываются к единому централизованному хранилищу *store*.

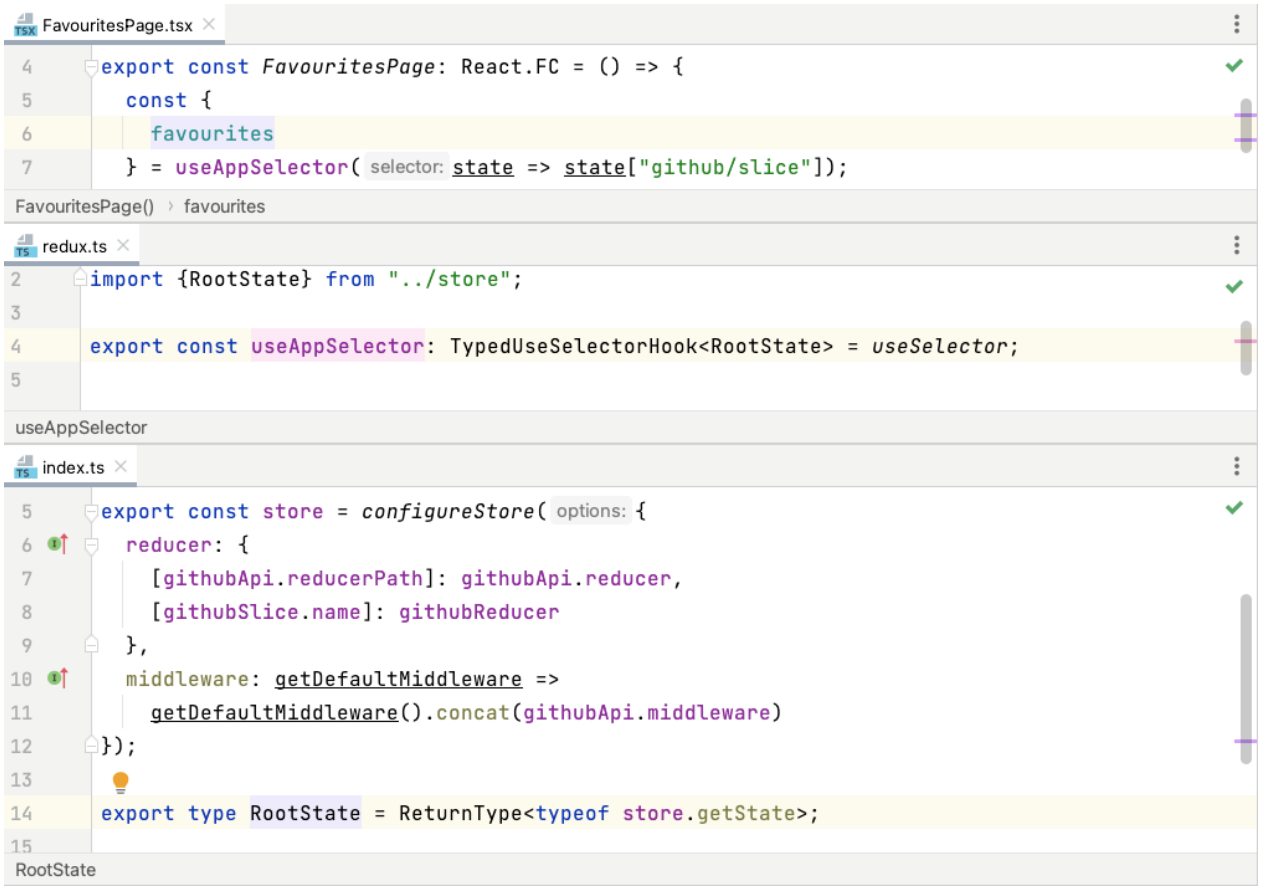

Рисунок 14 – Получение состояния среза по имени

Таким образом, был разобран стек современных технологий разработки на React.JS. Данный стек включает в себя продвинутое визуальное оформление страниц с помощью инструмента Tailwind CSS, который значительным образом ускоряет разработку. Также, была исследована библиотека Redux Toolkit, которая является надстройкой над классическим Redux и позволяет правильным и безызбыточным образом использовать все возможности данного инструмента, как

в контексте использования и реализации REST API, так и в контексте управления хранилищем в том числе с помощью срезов. Также, была в некоторой степени разобрана верстка страницы с использованием принципов React.JS на функциональных компонентах, что в итоге привело к конечному продукту, использующему в себе актуальные технологии.

Библиографический список

1. React.js [Электронный ресурс] URL: https://reactjs.com (дата обращения: 03.11.2022).
2. JavaScript | MDN [Электронный ресурс] URL: https://developer.mozilla.org/ru/docs/Web/JavaScript (дата обращения: 03.11.2022).
3. TypeScript [Электронный ресурс] URL: https://www.typescriptlang.org (дата обращения: 03.11.2022).
4. JetBrains IntelliJ IDEA [Электронный ресурс] URL: https://www.jetbrains.com/ru-ru/idea/ (дата обращения: 03.11.2022).
5. Node.JS [Электронный ресурс] URL: https://nodejs.org/en/ (дата обращения: 03.11.2022).
6. Create React App [Электронный ресурс] URL: https://create-react-app.dev (дата обращения: 03.11.2022).
7. Tailwind CSS [Электронный ресурс] URL: https://tailwindcss.com (дата обращения: 03.11.2022).
8. Bootstrap RUS [Электронный ресурс] URL: https://getbootstrap.ru (дата обращения: 03.11.2022).
9. PurgeCSS [Электронный ресурс] URL: https://purgecss.com (дата обращения: 03.11.2022).
10. React Router [Электронный ресурс] URL: https://reactrouter.com/en/main (дата обращения: 03.11.2022).

11. Redux [Электронный ресурс] URL: https://redux.js.org (дата обращения: 03.11.2022).
12. Redux Toolkit [Электронный ресурс] URL: https://redux-toolkit.js.org (дата обращения: 03.11.2022).
13. Immer: новый подход к иммутабельности в JavaScript [Электронный ресурс] URL: https://habr.com/ru/company/ruvds/blog/349492/ (дата обращения: 03.11.2022).
14. Redux Toolkit Query [Электронный ресурс] URL: https://redux-toolkit.js.org/rtk-query/overview (дата обращения: 03.11.2022).
15. json2ts [Электронный ресурс] URL: http://json2ts.com (дата обращения: 03.11.2022).

Bibliographic list

1. React.js [Electronic resource] URL: https://reactjs.com (accessed: 03.11.2022).
2. JavaScript | MDN [Electronic resource] URL: https://developer.mozilla.org/ru/docs/Web/JavaScript (accessed: 03.11.2022).
3. TypeScript [Electronic resource] URL: https://www.typescriptlang.org (accessed: 03.11.2022).
4. JetBrains IntelliJ IDEA [Electronic resource] URL: https://www.jetbrains.com/ru-ru/idea / (accessed: 03.11.2022).
5. Node.JS [Electronic resource] URL: https://nodejs.org/en / (accessed: 03.11.2022).
6. Create React App [Electronic resource] URL: https://create-react-app.dev (accessed: 03.11.2022).
7. Tailwind CSS [Electronic resource] URL: https://tailwindcss.com (accessed: 03.11.2022).
8. Bootstrap RUS [Electronic resource] URL: https://getbootstrap.ru (accessed: 03.11.2022).
9. PurgeCSS [Electronic resource] URL: https://purgecss.com (accessed: 03.11.2022).

10. React Router [Electronic resource] URL: https://reactrouter.com/en/main (accessed: 03.11.2022).
11. Redux [Electronic resource] URL: https://redux.js.org (accessed: 03.11.2022).
12. Redux Toolkit [Electronic resource] URL: https://redux-toolkit.js.org (accessed: 03.11.2022).
13. Immer: a new approach to immunability in JavaScript [Electronic resource] URL: https://habr.com/ru/company/ruvds/blog/349492 / (accessed: 03.11.2022).
14. Redux Toolkit Query [Electronic resource] URL: https://redux-toolkit.js.org/rtk-query/overview (accessed: 03.11.2022).
15. json2ts [Electronic resource] URL: http://json2ts.com (accessed: 03.11.2022).

Для цитирования: Берьянов М.С., Салахов И.Р., Иванов М.Д. ИССЛЕДОВАНИЕ СОВРЕМЕННОГО СТЕКА REACT РАЗРАБОТКИ В 2022 ГОДУ// Научный сетевой журнал «Столыпинский вестник» №8/2022.