Разработка интернет-приложений

Лабораторная работа 6

Библиотека React. Маршрутизация. Хук useParams.

4 часа

Цель работы: изучение библиотеки React и получение практических навыков при работе с ней.

Задачи:

- изучить основы работы системы маршрутизации при использовании библиотеки React;

- изучить механизм работы хука useParams при использовании библиотеки React;

- согласно варианту выполнить задание.

Теоретическая часть

Определение маршрутов

В React имеется своя система маршрутизации, которая позволяет сопоставлять запросы к приложению с определенными компонентами. Ключевым звеном в работе маршрутизации является модуль **react-router**, который содержит основной функционал по работе с маршрутизацией. Однако если мы собираемся работать в браузере, то нам также надо использовать модуль **react-router-dom**, а также **history**.

Библиотека react-router

Эта библиотека популярна, довольна проста в использовании и обладает хорошей документацией. Она предоставляет такие возможности как:

- Навигация по клику (компонент <Link>)
- Перенаправление (компонент <Redirect>)
- Маршрутизация (компонент Route)
- История (свойство history)

Прежде всего для работы с маршрутами необходимо добавить ссылки на **модули react-router-dom, react-router** и **history:**

```
<script src="https://unpkg.com/history@5/umd/history.development.js" cros-
sorigin></script>
<script src="https://unpkg.com/react-router@6/umd/react-router.develop-
ment.js" crossorigin></script>
<script src="https://unpkg.com/react-router-dom@6/umd/react-router-dom.de-
velopment.js" crossorigin></script>
```

В системе маршрутизации каждый маршрут сопоставляется с определенным компонентом, поэтому для примера определим три однотипных компонента: Main, About и NotFound:

```
class About extends React.Component{
    render(){
        return <h2>О сайте</h2>;
    }
}
class NotFound extends React.Component{
    render(){
        return <h2>Ресурс не найден</h2>;
    }
}

class Main extends React.Component{
    render(){
        return <h2>Главная</h2>;
    }
}
```

В начале для работы с маршрутами также получаем ряд объектов, которые потребуются для определения маршрутов:

```
const Router = ReactRouterDOM.BrowserRouter;
const Route = ReactRouterDOM.Route;
const Routes = ReactRouterDOM.Routes;
```

Здесь определены три объекта из модуля **react-router-dom**.

Router определяет набор маршрутов и, когда к приложению, приходит запрос, то Router выполняет сопоставление запроса с маршрутами. И если какой-то маршрут совпадает с URL запроса, то этот маршрут выбирается для обработки запроса.

И также для выбора маршрута определен объект **Routes**. Он содержит набор маршрутов и позволяет выбрать первый попавшийся маршрут и его использовать для обработки.

Каждый маршрут представляет объект **Route**. Он имеет ряд атрибутов. В частности, здесь для маршрута устанавливаются два атрибута:

- **path:** шаблон адреса, с которым будет сопоставляться запрошенный адрес URL;
- **element** - тот компонент, который отвечает за обработку запроса по этому маршруту.

Например, первый маршрут выступает в качестве корневого. Он сопоставляется с адресом "/" и обрабатывается компонентом Main:

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

Особо следует выделить третий маршрут:

```
<Route path="*" element={<NotFound />} />
```

Путь в виде звездочки - "*" указывает, что этот маршрут будет сопоставляться со всеми адресами URL, которые не соответствуют предыдущим маршрутам. И он будет

обрабатываться компонентом **NotFound**. Таким образом мы можем задать обработку при обращении к несуществующим ресурсам в приложении.

При работе с маршрутами следует учитывать, что мы не сможем просто кинуть страницу **index.html** в браузер, и у нас все заработает, как в прошлых статьях. Чтобы система маршрутизации заработала, нам надо разместить файл index.html на веб-сервере. В качестве веб-сервера можно использовать любой понравивший веб-сервер (Apache, IIS, Ngnix и т.д.) или обращаться к данной html-странице в рамках веб-приложения. В данном же случае я буду использовать Node.js как самый демократичный и распространенный вариант.

Дочерние маршруты

В рамках маршрутов в React можно определять дочерние маршруты. Такие подмаршруты будут отсчитываться от главного маршрута. Но для построения подобной системы есть ряд подходов. Рассмотрим их.

Определение подмаршрутов в коде компонента

Для применения подмаршрутов возьмем из прошлого примера 3 компонента: Main, About и NotFound и добавим к ним еще компонент Products:

```
class Products extends React.Component{
        render(){
            return <div>
                        <h2>Товары</h2>
                        <Routes>
                            <Route path="/phones" element={<Phone />} />
//вложенный маршрут
                            <Route path="/tablets" element={<Tablet />} />
//вложенный маршрут

                        </Routes>
                    </div>;
        }
    }
```

Для обработки запроса "/products" здесь определен маршрут, который обрабатывается компонентом Products:

```
<Route path="/products/*" element={<Products />} />
```

Обратите внимание на шаблон пути: path="/products/*". Символ * указывает, что компонент Products будет обрабатывать маршруты, которые начинаются "/products/", но после слеша также могут идти и другие символы.

Но сам этот компонент имеет вложенные маршруты, как мы видим выше.

Вложенные маршруты отсчитываются фактически от главного маршрута "/products". То есть маршрут

```
<Route path="/phones" element={<Phone />} />
```

будет обрабатывать запросы по пути "/phones", который добавляется к пути главного компонента - "/products", то есть в итоге по пути "/products/phones". Аналогичным образом запросы по пути "/products/tablets" будут обрабатываться компонентом Tablet.

И что также можно заметить, то в обоих случаях выводится заголовок "Товары", так как он определен на главном компоненте Products. Остальное содержимое будет отличаться в зависимости от выбранного для обработки запроса компонента.

Однако данный подход имеет как минимум один недостаток - при обращении к любым адресам, которые начинаются с "products", запросы будет обрабатывать компонент Products. Но, к примеру, мы хотим, чтобы он обрабатывал свой основной маршрут и запросы по дочерним маршутам. Поэтому рассмотрим другой подход.

Outlet

При другом подходе маршруты определяются вне компонента, а их содержимое вставляется в главный компонент с помощью встроенного компонента Outlet. Так, изменим предыдущий пример следующим образом:

```
class Products extends React.Component{
        render(){
            return <div>
                        <h2>Товары</h2>
                        <Outlet />
                    </div>;
        }
    }
```

Прежле всего в коде компонента Products применяется компонент Outlet. Здесь вместо элемента <Outlet /> будет вставляться содержимое компонентов, которые обрабатывают дочерние маршруты.

Также определим дочерние маршруты внутри основного маршрута:

```
<Route path="/products" element={<Products />}>
    <Route path="phones" element={<Phone />} />
    <Route path="tablets" element={<Tablet />} />
</Route>
```

Стоит отметить, что пути в дочерних маршрутах не должны начинаться со слеша. И в итоге дочерние маршруты также будут отсчитываются от главного маршрута "/products" и сопвадать с запросами "/products/phones" и "/products/tablets". Запрос по основному маршруту "/products" будет обрабатываться только компонентом Products. Все остальные запросы, которые не совпадают с основным и подмаршрутами, например, "/products/abc", будет обрабатываться самым последним маршрутом.

Создание ссылок

Добавим к прошлому примеру навигацию в виде ссылок.

Для создания ссылки применяется объект Link, который определен в модуле react-router-dom:

```
const Link = ReactRouterDOM.Link;
```

Для определения блока навигации добавим компонент Nav:

```
class Nav extends React.Component{
    render(){
        return <nav>
                <Link to="/">Главная</Link>
                <Link to="/about">О сайте</Link>
                <Link to="/products">Товары</Link>
               </nav>;
    }
}
```

Для каждой ссылки с помощью атрибута **to** определяет путь перехода.

Затем компонент Nav помещается в объект Router. И после запуска мы увидим блок ссылок, по которым сможем переходить к ресурсам приложения.

Кроме объекта Link из модуля **react-router-dom** для создания ссылок мы можем использовать объект **NavLink**. Этот объект во многом аналогичен Link за тем исключением, что позволяет использовать состояние ссылки. В частности, с помощью атрибутов className и style можно установить стиль активной ссылки.

Параметры маршрутов

Маршруты могут принимать параметры. Параметр имеет следующую форму определения: `:название_параметра`.

Для получения параметров маршрута применяется встроенная функция-хук **useParams**, соответственно ее вначале необходимо получить:

```
const useParams = ReactRouterDOM.useParams;
```

Далее определяем один маршрут, который принимает параметр:

```
<Route path=":id" element={<Product />} />
```

Причем он определен как дочерний по отношению к маршруту "/products". Этот маршрут будет обрабатываться компонентом Product. Причем надо учитывать, что хуки могут применяться только в функциональных компонентах. Поэтому компонент Product определен в виде функции:

```
function Product(){
    // получаем параметры
    const params = useParams();
    const prodId = params.id;
    return <h2>Товар № {prodId}</h2>;
}
```

Функция **useParams()** возвращает набор параметров маршрута. Фактически такой набор представляет объект, а каждый отдельный параметр - свойство этого объекта. И чтобы получить параметр id, необходимо использовать выражение params.id.

Если же параметр не будет передан, например, при запросе http://localhost:3000/products, то он будет обрабатываться компонентом ProductsList.

В итоге мы получаем следующий код:

```
const Router = ReactRouterDOM.BrowserRouter;
const Route = ReactRouterDOM.Route;
const Routes = ReactRouterDOM.Routes;
const useParams = ReactRouterDOM.useParams;
const Outlet = ReactRouterDOM.Outlet;

class ProductsList extends React.Component{
    render(){
        return <h2>Список товаров</h2>;
    }
}
function Product(){
    // получаем параметры
    const params = useParams();
    const prodId = params.id;
    return <h2>Товар № {prodId}</h2>;
}
class Products extends React.Component{
    render(){
        return (<div>
            <h1>Товары</h1>
            <Outlet />
        </div>)
    }
}
ReactDOM.render(
    <Router>
        <div>
          <Routes>
            <Route path="/" element={<h2>Главная</h2>} />
            <Route path="/products" element={<Products />}>
                <Route index element={<ProductsList />} />
                <Route path=":id" element={<Product />} />
            </Route>
            <Route path="*" element={<h2>Ресурс не найден</h2>} />
          </Routes>
        </div>
    </Router>,
    document.getElementById("app")
)
```

Ссылки с параметрами

Если мы хотим добавить к адресу ссылки параметр, то весь адрес помещается в фигурные скобки. В прошлый пример добавим дополнительный массив для эмуляции хранилища данных:

```
const phones =[
                    {id: 1, name: "Apple iPhone 12 Pro"},
                    {id: 2, name: "Google Pixel 5"},
                    {id: 3, name: "Huawei P40 Pro"}
                ];
```

Для доступа к данным этого массива выводятся ссылки, которые определены в компоненте ProductList:

```
class ProductsList extends React.Component{
        render(){
            return <div>
                    <h2>Список товаров</h2>
                    <ul>
                    {
                        phones.map(function(item){
                            return <li key={item.id}>
                                      <NavLink to={`/prod-
ucts/${item.id}`}>{item.name}</NavLink> //здесь выводятся ссылки
                                   </li>
                        })
                    }
                </ul>
            </div>;
        }
    }
```

И при переходе по адресу "/products" мы увидим набор ссылок. При переходе по ссылке компонент Product получит значение параметра и по нему извлекет из массива phones нужный объект.

Задание на лабораторную работу

1) Создать веб-сайт согласно варианту, который соответствует следующим критериям:
    - содержит минимум 4 кликабельных пункта меню (каждый пункт является ссылкой);
    - 2 пункта меню имеют вложенное меню (подменю), каждый пункт подменю является ссылкой;
    - 1 пункт подменю содержит компонент списка, при нажатии на элемент списка, открывается страница с дополнительной информацией по выбранному элементу (например, текст, картинка);
    - использовать массив для эмуляции хранилища данных;
    - оформление сайта по вашему усмотрению (может быть совсем простым), главное показать навык работы с маршрутизацией и использования хука **useParams().**

Вариант	Тема
1	Магазин книг
2	Магазин сотовых телефонов
3	Магазин курсов по программированию
4	Магазин продуктов
5	Магазин обуви
6	Магазин зоотоваров