Маршрутизация

ПОДГОТОВКА

Для данного занятия нам потребуется статический web сервер. Установить его можно, например, следующим образом:

- открыть консоль (терминал) в новой пустой папке
- выполнить команду: npm i express
- создать папку для статического контента с названием public и в него положить тестовый index.html
- затем рядом с папкой создать файл app.js со следующим кодом:

```
const express = require('express');
const app = express();
app.use(express.static('public'));
app.listen(8080);
```

- переходим в браузер по ссылке: http://localhost:8080/

БИБЛИОТЕКА VUE-ROUTER

Vue.js имеет полноценную систему маршрутизации, которая позволяет сопоставлять запросы к приложению с определенными компонентами. За работу системы маршрутизации во Vue.js отвечает специальная библиотека - **vue-router**.

Подключение библиотеки:

```
<script src="https://unpkg.com/vue-router/dist/vue-router.js"></script>
```

ОПРЕДЕЛЕНИЕ МАРШРУТОВ

В шаблоне объекта Vue помещается компонент <router-view>:

```
<div id="app">
    <router-view></router-view>
</div>
```

В этот элемент будет помещаться выбранный для обработки запроса компонент.

Создадим несколько компонентов:

```
// компоненты, которые обрабатывают пути
const NotFound = { template: '<h2>Page Not Found</h2>' };
const Home = { template: '<h2>Home Page</h2>' };
const About = { template: '<h2>About Page</h2>' };
```

ОПРЕДЕЛЕНИЕ МАРШРУТОВ

Маршруты определяются путем создания массива с описанием маршрутов:

```
// маршруты, сопоставленные с компонентами
const routes = [
  { path: '/', component: Home },
  { path: '/about', component: About },
  { path: '*', component: NotFound }
];
```

Каждый маршрут определяет свойство path, которое представляет путь запроса, и свойство component - компонент, который будет обрабатывать запрос по этому пути.

Таким образом, компонент Home будет обрабатывать запрос по пути "/", то есть по сути запрос к корню сайта. Компонент About обрабатывает запросы по пути "/about". А компонент NotFound будет обрабатывать все остальные пути, для этого для него свойство path имеет в качестве значения "*". Причем когда приложение получит запрос, то этот запрос будет последовательно сопоставляться со всеми маршрутами. Первый маршрут, у которого свойство path совпадет с путем запроса и будет выбран для обработки.

ОПРЕДЕЛЕНИЕ МАРШРУТОВ

Создаём экземпляр маршрутизатора:

```
const router = new VueRouter({
    /*mode: 'history',*/
    routes: routes
});
```

У этого объекта устанавливаются два свойства. Свойство mode указывает на используемый режим навигации. В частности, значение "history" применяет history.pushState API, которое позволяет использовать навигацию без перезагрузки страницы.

Далее свойство routes устанавливает маршруты - это выше определенные маршруты.

ОПРЕДЕЛЕНИЕ МАРШРУТОВ

Создаём и монтируем корневой экземпляр приложения:

```
const app = new Vue({
  router: router
}).$mount('#app');
```

Передаем экземпляр маршрутизатора в опции router, чтобы позволить приложению знать о его наличии.

Если при создании экземпляра Vue не была указана опция el, он окажется в «непримонтированном» (unmounted) состоянии, то есть без ассоциированного элемента DOM. Вызовите vm.$mount(), чтобы примонтировать такой экземпляр.

НАВИГАЦИЯ И ССЫЛКИ

Для создания системы навигации в приложении Vue.js применяется компонент router-link. С помощью свойства to у данного компонента можно установить путь для создаваемой ссылки.

```
<div id="app">
    <ul>
        <li><router-link to="/" exact>Home</router-link></li>
        <li><router-link to="/products">Products</router-link></li>
        <li><router-link to="/about">About</router-link></li>
    </ul>
    <router-view></router-view>
</div>
```

У первой ссылки указан атрибут exact. Дело в том, что путь "/" фактически соответствует любому пути, который начинается с этого символа, в том числе и путям "/about" и "/products". Атрибут exact обеспечивает точное соответствие маршрута только с путем "/". Стоит отметить, что при нажатии на ссылку к элементу будет добавляться класс router-link-active. И мы можем использовать это обстоятельство для стилизации активной ссылки.

НАВИГАЦИЯ И ССЫЛКИ

Дополнительно мы можем установить другие свойства для router-link. Например, определим новый класс CSS:.

```
.active {
  color: green;
}
```

Чтобы этот класс использовался в качестве класса для активной ссылки, необходимо использовать свойство active-class:

```
<ul>
    <li><router-link to="/" exact tag="div"
            active-class="active">Home</router-link></li>
    <li><router-link to="/products" tag="div"
            active-class="active">Products</router-link></li>
    <li><router-link to="/about" tag="div"
            active-class="active">About</router-link></li>
</ul>
```

С помощью атрибута tag можно установить, в какой элемент будет помещаться ссылка.

ПАРАМЕТРЫ МАРШРУТОВ

Определяемые во Vue.js маршруты могут содержать параметры. Для определения параметра в конец маршрута добавляется двоеточие, после которого идет название параметра:

```
{ path: '/products/:id', component: Products }
```

Например, в данном случае определен параметр id. И такому маршруту , к примеру, будут соответствовать такие пути запроса как:
/products/6-tom
/products/21
/products/phones

Та часть, которая идет после последнего слеша, будет интерпретироваться как значение параметра id. В компоненте Vue.js мы можем получить параметры запроса через объект $route.params, который содержит значения всех параметров. В частности, для обращения к параметру id нужно использовать выражение $route.params.id.

ПАРАМЕТРЫ МАРШРУТОВ

Пример:

```
<div id="app">
    <ul>
      <li><router-link to="/" exact>Home</router-link></li>
      <li><router-link to="/products/1">Product 1
          </router-link></li>
      <li><router-link to="/products/2">Product 2
          </router-link></li>
    </ul>
    <router-view></router-view>
</div>
<script>

    const NotFound = { template: '<h2>Page Not Found</h2>'}
    const Home = { template: '<h2>Home Page</h2>' }
    const Products = { template: '<h2>Product
{{$route.params.id }}</h2>' }
```

ПАРАМЕТРЫ МАРШРУТОВ

Пример (продолжение):

```
    const routes = [
      { path: '/', component: Home },
      { path: '/products/:id', component: Products },
      { path: '*', component: NotFound }
    ];
    // создаем объект маршрутизатора
    const router = new VueRouter({
        mode: 'history',
        routes: routes
    });

    const app = new Vue({
        router: router
    }).$mount('#app');
</script>
```

ПАРАМЕТРЫ МАРШРУТОВ

Подобным образом можно использовать большее количество параметров:

```
<div id="app">
    <ul>
      <li><router-link to="/" exact>Home</router-link></li>
      <li><router-link to="/products/tablets/2">Tablet 2
          </router-link></li>
      <li><router-link to="/products/phones/3">Phone 3
          </router-link></li>
    </ul>
    <router-view></router-view>
</div>
<script>

    const NotFound = { template: '<h2>Page Not Found</h2>'}
    const Home = { template: '<h2>Home Page</h2>' }
     const Products = { template: `<div><h2>Product</h2>
        <h3>Category: {{$route.params.category }}</h3>
        <h3>Id: {{$route.params.id }}</h3>
     </div>`
    }
```

ПАРАМЕТРЫ МАРШРУТОВ

Подобным образом можно использовать большее количество параметров (продолжение):

```
    const routes = [
      { path: '/', component: Home },
      { path: '/products/:category/:id',
        component: Products },
      { path: '*', component: NotFound }
    ];
    // создаем объект маршрутизатора
    const router = new VueRouter({
        mode: 'history',
        routes: routes
    });

    const app = new Vue({
        router: router
    }).$mount('#app');
</script>
```

ПАРАМЕТРЫ МАРШРУТОВ

Параметры могут быть необязательными. Для таких параметров необязательно передавать значения. Для их определения после названия параметра ставится вопросительный знак:

```
<div id="app">
    <ul>
      <li><router-link to="/" exact>Home</router-link></li>
      <li><router-link to="/products/1">Product 1
          </router-link></li>
      <li><router-link to="/products/">Products
          </router-link></li>
    </ul>
    <router-view></router-view>
</div>
<script>
    const NotFound = { template: '<h2>Page Not Found</h2>'}
    const Home = { template: '<h2>Home Page</h2>' }
    const Products = { template: `<div>
        <h2 v-if="$route.params.id">Товар {{$route.params.id}}</h2>
        <h2 v-else>Список товаров</h2>
      </div>`
    }
```

ПАРАМЕТРЫ МАРШРУТОВ

Пример (продолжение):

```
    const routes = [
      { path: '/', component: Home },
      { path: '/products/:id?', component: Products },
      { path: '*', component:NotFound }
    ];
    // создаем объект маршрутизатора
    const router = new VueRouter({
        mode: 'history',
        routes: routes
    });

    const app = new Vue({
        router: router
    }).$mount('#app');
</script>
```

ВЛОЖЕННЫЕ МАРШРУТЫ

Во Vue.js одни маршруты могут быть вложенными в другие, то есть подмаршрутами:

```
<div id="app">
    <ul>
      <li><router-link to="/" exact>Home</router-link></li>
      <li><router-link to="/products/phones">Смартфоны
          </router-link></li>
      <li><router-link to="/products/tablets">Планшеты
          </router-link></li>
    </ul>
    <router-view></router-view>
</div>
<script>
    const NotFound = { template: '<h1>Страница не найдена</h1>' }
    const Home = { template: '<h1>Главная</h1>' }

    const Phones = { template: '<h2>Смартфоны</h2>' }
    const Tablets = { template: '<h2>Планшеты</h2>' }
    const Products = { template: '<div><h1>Товары</h1><router-
view></router-view></div>' }
```

ВЛОЖЕННЫЕ МАРШРУТЫ

Пример (продолжение):

```
      const routes = [
      { path: '/', component: Home },
      {
        path: '/products',
        component: Products,
        children: [ { path: 'phones', component: Phones },
          { path: 'tablets', component: Tablets }]
        },
        { path: '*', component: NotFound }
    ];
    const router = new VueRouter({
        mode: 'history',
        routes: routes
    });

    const app = new Vue({
        router: router
    }).$mount('#app');
</script>
```

ВЛОЖЕННЫЕ МАРШРУТЫ

Также можно задать подмаршрут, который будет обрабатывать запросы к корню родительского маршрута:

```
<div id="app">
    <ul>
      <li><router-link to="/" exact>Home</router-link></li>
      <li><router-link to="/products">Товары</router-link></li>
    </ul>
    <router-view></router-view>
</div>
<script>
    const NotFound = { template: '<h1>Страница не найдена</h1>' }
    const Home = { template: '<h1>Главная</h1>' }
    const Phones = { template: '<h2>Смартфоны</h2>' }
    const Tablets = { template: '<h2>Планшеты</h2>' }
    const Index = { template: `<div>
        <h3>Выберите категорию</h3>
        <div><router-link to="/products/phones">Смартфоны</router-link></div>
        <div><router-link to="/products/tablets">Планшеты</router-link></div>
      </div>`
    }
    const Products = { template: '<div><h1>Товары</h1><router-view></router-
view></div>' }
```

ВЛОЖЕННЫЕ МАРШРУТЫ

Пример (продолжение):

```
    const routes = [
      { path: '/', component: Home },
      {
        path: '/products',
        component: Products,
        children: [ { path: 'phones', component: Phones },
          { path: 'tablets', component: Tablets },
          { path: '', component: Index }]
       },
       { path: '*', component: NotFound }
    ];
    const router = new VueRouter({
        mode: 'history',
        routes: routes
    });

    const app = new Vue({
        router: router
    }).$mount('#app');
</script>
```

ИМЕНОВАННЫЕ МАРШРУТЫ

Маршруты во Vue.js могут иметь имена. Используя имена маршрутов в дальнейшем мы можем, к примеру, привязывать пути ссылок к этим маршрутам:

```
<div id="app">
        <div><router-link to="/" exact>Home</router-link></div>
        <router-view></router-view>
</div>
<script>
 const NotFound = { template: '<h2>Страница не найдена</h2>' }
    const Home = { template: `<div><h2>Главная</h2>
      <ul>
        <li><router-link :to="{ name:'products', params:{ id: 1 }}">
Товар 1</router-link></li>
        <li><router-link :to="{ name:'products', params:{ id: 2 }}">
Товар 2</router-link></li>
        <li><router-link :to="{ name:'products', params:{ id: 3 }}">
Товар 3</router-link></li>
        <li><router-link :to="{ name:'products', params:{ id: 4 }}">
Товар 4</router-link></li>
      </ul></div>` }
    const Products = { template: '<h2>Товар
{{$route.params.id}}</h2>' }
```

ИМЕНОВАННЫЕ МАРШРУТЫ

Пример (продолжение):

```
    const routes = [
      { path: '/', component: Home },
      {
        path: '/products/:id',
        component: Products,
        name: 'products'
        },
        { path: '*', component: NotFound }
    ];

    const router = new VueRouter({
        mode: 'history',
        routes: routes
    });

    const app = new Vue({
        router: router
    }).$mount('#app');
</script>
```

ИМЕНОВАННЫЕ ПРЕДСТАВЛЕНИЯ

Для создания сложных макетов мы можем использовать в одном шаблоне сразу несколько элементов <router-view> (представлений). При этом для каждого представления можно определить свое имя с помощью атрибута name. Затем, используя данное имя мы сможем сопоставить это представление с определенным компонентом, который будет определять содержимое представления:

```
<div id="app">
    <router-view></router-view>
    <router-view name="ads"></router-view>
    <router-view name="content"></router-view>
<script>
    const NotFound = { template: '<h2>Страница не найдена</h2>' }
    const HomeLink = { template: '<router-link to="/" exact> Главная</router-
link>' }
    const Products = { template: `
      <div>
      <h2>Товары</h2>
        <ul>
          <li><router-link to="/phones">Смартфоны</router-link></li>
          <li><router-link to="/tablets">Планшеты</router-link></li>
        </ul>
      </div>`
    }
```

ИМЕНОВАННЫЕ ПРЕДСТАВЛЕНИЯ

Пример (продолжение):

```
const PhonesAds = { template: '<div><h3>Реклама смартфонов</h3>
   <p>Покупайте смартфоны фирмы Y</p></div>' }
const TabletsAds = { template: '<div><h3>Реклама планшетов</h3>
   <p>Покупайте планшеты фирмы X!</p></div>' }

const Phones ={template:'<h2>Все о смартфонах</h2>'}
const Tablets ={template:'<h2>Все о планшетах</h2>'}

const routes = [
  { path: '/',
    components: { default: HomeLink, content: Products }
  },
  {
    path: '/phones',
    components: { default: HomeLink, ads: PhonesAds, content: Phones }
  },
  {
    path: '/tablets',
    components: { default: HomeLink, ads: TabletsAds, content: Tablets }
  },
  { path: '*', component: NotFound }
];
```

ИМЕНОВАННЫЕ ПРЕДСТАВЛЕНИЯ

Пример (продолжение):

```
    const router = new VueRouter({
        mode: 'history',
        routes: routes
    });

    const app = new Vue({
        router: router
    }).$mount('#app');
</script>
```

ПЕРЕАДРЕСАЦИЯ

В процессе работы приложения может сложиться ситуация, что какой-то ресурс будет перемещен с одного адреса на другой. В этом случае возникает необходимость установить переадресацию с одного адреса на другой. Для выполнения переадресации у маршрута применяется параметр redirect, который задает путь для редиректа:

```
<div id="app">
    <router-link to="/404">404 страница</router-link>
    <router-link to="/some">Какая-то страница</router-link>
    <router-view></router-view>
<script>
    const Home = { template: '<h2>Главная</h2>' }
    const NotFound = { template: '<h2>Страница не найдена</h2>' }
    const routes = [
      { path: '/', component: Home },
      { path: '/404', component:NotFound },
      { path: '*', redirect: "/404" }
    ];

    const router = new VueRouter({ mode: 'history', routes: routes });

    const app = new Vue({
        router: router
    }).$mount('#app');
</script>
```

ПЕРЕАДРЕСАЦИЯ

При переадресации можно передавать параметры:

```
<div id="app">
    <router-link to="/product/1">Товар 1</router-link>
    <router-link to="/product/2">Товар 2</router-link>
    <router-view></router-view>
<script>
    const Home = { template: '<h2>Главная</h2>' }
    const Product = { template: '<h2>Товар {{$route.params.id}}</h2>' }
    const NotFound = { template: '<h2>Страница не найдена</h2>' }

    const routes = [
      { path: '/', component: Home },
      { path: '/product/:id', redirect:'/ru/product/:id' },
      { path: '/ru/product/:id', component: Product },
      { path: '/404', component:NotFound },
      { path: '*', redirect: "/404" }
    ];

    const router = new VueRouter({ mode: 'history', routes: routes });

    const app = new Vue({
        router: router
    }).$mount('#app');
</script>
```

ПЕРЕАДРЕСАЦИЯ

Чтобы проще было управлять переадресацией можно установить редирект на маршрут по определенному имени:

```
<div id="app">
    <router-link to="/product/1">Товар 1</router-link>
    <router-link to="/product/2">Товар 2</router-link>
    <router-view></router-view>
<script>
    const Home = { template: '<h2>Главная</h2>' }
    const Product = { template: '<h2>Товар {{$route.params.id}}</h2>' }
    const NotFound = { template: '<h2>Страница не найдена</h2>' }

    const routes = [
      { path: '/', component: Home },
      { path: '/product/:id', redirect: {name: 'product'} },
      { path: '/ru/product/:id', component: Product, name:'product' },
      { path: '/404', component:NotFound },
      { path: '*', redirect: "/404" }
    ];
    const router = new VueRouter({ mode: 'history', routes: routes });
    const app = new Vue({
        router: router
    }).$mount('#app');
</script>
```

ПЕРЕАДРЕСАЦИЯ

Иногда возникает ситуация, когда надо динамически решить, куда выполнять переадресацию. В этом случае можно задать функцию, которая динамически определяет, куда переадресовывать пользователя:

```
<div id="app">
    <router-link to="/product/1">Товар 1</router-link>
    <router-link to="/product/20">Товар 20</router-link>
    <router-view></router-view>
<script>
    const Home = { template: '<h2>Главная</h2>' }
    const Product = { template: '<h2>Товар {{$route.params.id}}</h2>' }
    const NotFound = { template: '<h2>Страница не найдена</h2>' }
```

ПЕРЕАДРЕСАЦИЯ

Пример (продолжение):

```
 const routes = [
      { path: '/', component: Home },
      { path: '/product/:id', redirect: to => {
         if (to.params.id > 10) {
            return '/404'
          } else {
            return '/ru/product/:id'
          }
        }
      },
      { path: '/ru/product/:id', component: Product, name:'product' },
      { path: '/404', component:NotFound },
      { path: '*', redirect: "/404" }
    ];

    const router = new VueRouter({
        mode: 'history',
        routes: routes
    });
    const app = new Vue({
        router: router
    }).$mount('#app');
</script>
```

ПРОГРАММНАЯ НАВИГАЦИЯ

Помимо декларативного использования <router-link> для создания ссылок в шаблоне, можно использовать маршрутизатор и программно вызывать методы его экземпляра.

Декларативная запись	Программная запись
`<router-link :to="...">`	`router.push(...)`

```
// строка
router.push('home');
// объект
router.push({ path: 'home' });
// именованный маршрут
router.push({ name: 'user', params: { userId: '123' } });
// со строкой запроса, получится /register?plan=private
router.push({ path: 'register/1'});
```

params игнорируются, если указан path. Вместо этого, вам нужно указать name маршрута или вручную указать весь path с необходимыми параметрами:

```
const userId = '123'
router.push({ name: 'user', params: { userId } }) // -> /user/123
router.push({ path: `/user/${userId}` })
```

Внутри экземпляра Vue у вас есть доступ к экземпляру маршрутизатора через $router. Поэтому вы можете вызвать this.$router.push

ДОПОЛНИТЕЛЬНЫЕ ССЫЛКИ

Официальное руководство по Vue.js:
https://ru.vuejs.org/v2/guide/

Руководство по Vue.js (онлайн учебник)
https://metanit.com/web/vuejs/

Официальное руководство по библиотеки Vue Router:
https://router.vuejs.org/ru/guide/