



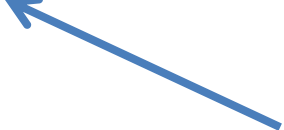
ANGULAR ROUTER



Параметры

Работа с параметрами url – еще одна гибкая возможность Angular.

```
const routes: Routes = [  
  {path: '', component: HomeComponent},  
  {path: 'about', component: AboutComponent},  
  {path: 'order-data/:id', component: OrderDataComponent},  
];
```



Параметр с именем «id».
Имя задает разработчик!

Рассмотрим пример, при котором пользователь имеет право ввести необходимый для перехода id.

Для этого модифицируем HomeComponent html и ts.

```
<h1>  
  home works!  
</h1>  
<h3>Please enter order id:</h3>  
<input [(ngModel)]="orderId">
```

```
<button (click)="navigateToOrder()">Navigate to order with id</button>
```

Параметры

```
export class HomeComponent {  
  
  orderId: number;  
  
  constructor(private router: Router) {  
  }  
  
  navigateToOrder() {  
    console.log(this.orderId);  
    this.router.navigate(commands: ['order-data', this.orderId]);  
  }  
  
}
```

infopulse
univer

[Home](#) About Order User

home works!

Please enter order id:

Navigate

infopulse
univer

[Home](#) About Order User

home works!

Please enter order id:

213

Navigate to order v

localhost:4200/order-data/213

Google Переводчик Codemotion Mail Зарп

infopulse
univer

Home About [Order](#) User

order-data works!

Введенный
параметр

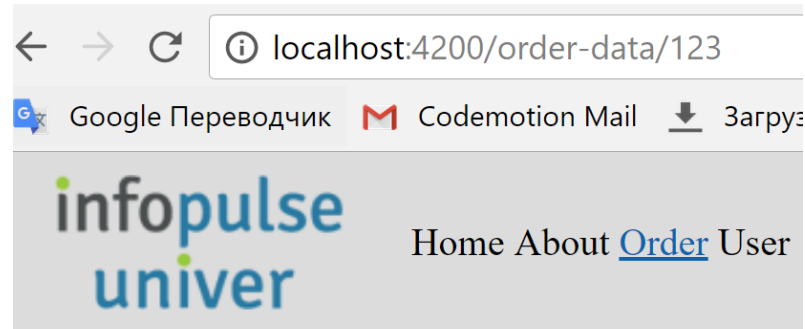
Работа с параметрами в коде

Класс **ActivatedRoute** – предоставляет возможность работать с параметрами url, внутри кода.

Модифицируем **OrderDataComponent**:

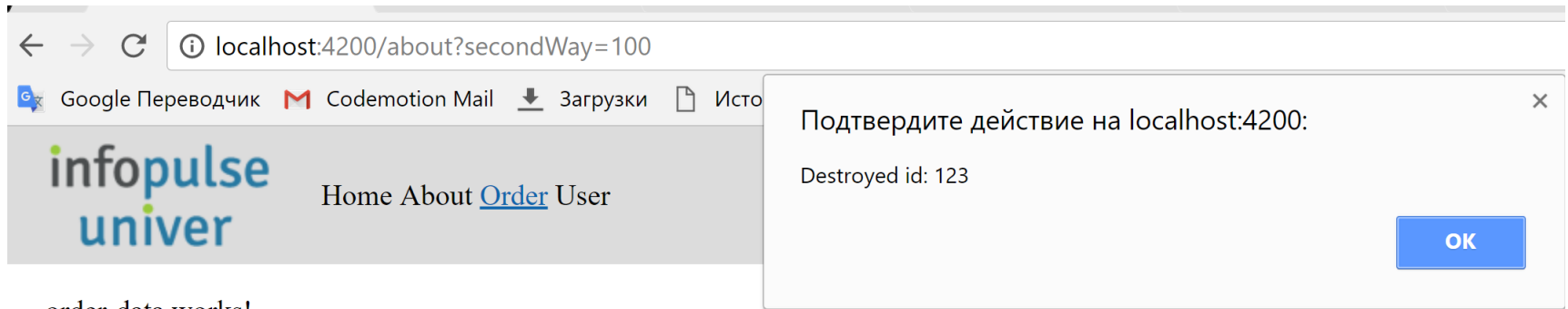
```
export class OrderDataComponent implements OnDestroy {  
  
  private currentId;  
  
  constructor(private activatedRoute: ActivatedRoute) {  
  
    this.activatedRoute.params.subscribe( next: (param: any) => {  
      this.currentId = param['id'];  
    });  
  }  
  
  ngOnDestroy() {  
    alert('Destroyed id: ' + this.currentId);  
  }  
  
}
```

Работа с параметрами в коде



order-data works!

Далее кликнем на ссылку **Home** или **About**.



order-data works!

Таким образом мы получили и обработали **параметр id**, через наш код.

Обратите внимание, что сработал **onDestroy()** – а это значит что **Angular уничтожает** не нужные в данный момент компоненты. Это позволяет ему работать **быстро и эффективно**.

Query Parameters

Иногда случаются ситуации, когда Вы не знаете будет существовать определенный параметр или нет.

В таких ситуациях **Angular** предоставляет возможность использования **queryParams**.

Важно понимать что в отличии от обычных параметров, query params **не нужно** указывать в файле роутинга.

Модифицируем **ts** и **html** в HomeComponent:

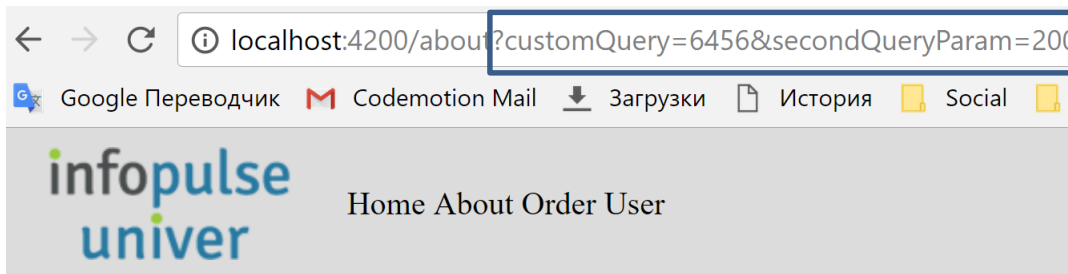
```
<h1>
  home works!
</h1>
<h3>Please enter order id:</h3>
<input [(ngModel)]="orderId">

<button (click)="navigateToOrder()">Navigate to order with id</button>
<p>
<button (click)="navigateToAbout()">Navigate to about</button>
</p>
```

Query Parameters

```
export class HomeComponent {  
  
  orderId: number;  
  
  constructor(private router: Router) {  
  }  
  
  navigateToOrder() {  
    this.router.navigate(commands: ['order-data', this.orderId]);  
  }  
  
  navigateToAbout() {  
    this.router.navigate(commands: ['about'], extras: {queryParams: {'customQuery': 6456, 'secondQueryParam': 200}});  
  }  
}
```

После нажатия кнопки «**Navigate to about**» в **HomeComponent**, мы перейдем на компонент **About** и увидим следующее:



**Query
Parameters**

about works!

Navigate to home

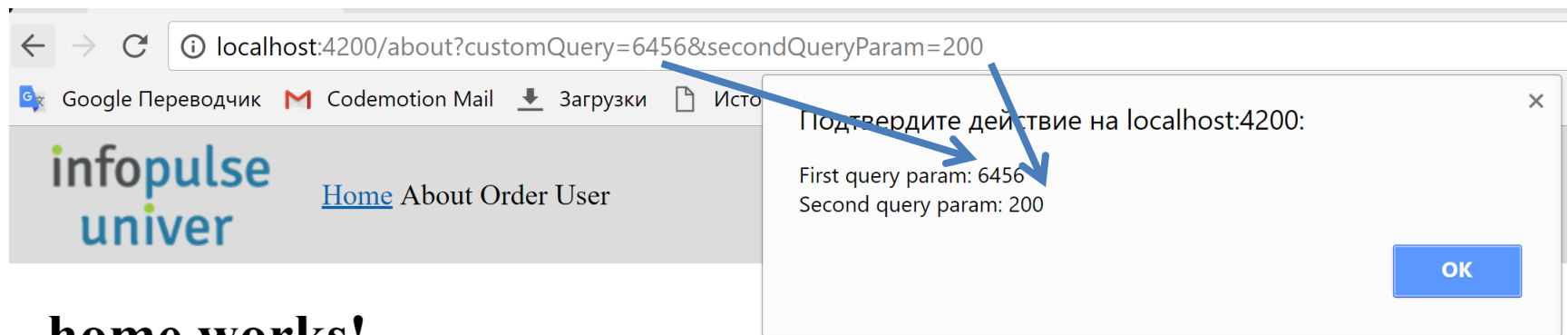
Работа с Query Parameters

Для взаимодействия с **query params** внутри кода, необходимо использовать уже известный сервис **ActivatedRoute**.

```
constructor(private router: Router, private activatedRoute: ActivatedRoute) {  
  
  this.activatedRoute.queryParams.subscribe(next: (queryParams: any) => {  
    this.firstQueryParam = queryParams['customQuery'];  
    this.secondQueryParam = queryParams['secondQueryParam'];  
  });  
  
}
```

Получим значение **query param**, после инициализации компонента:

```
ngOnInit() {  
  if ((this.firstQueryParam && this.secondQueryParam) != null)  
    alert('First query param: ' + this.firstQueryParam + '\n' + 'Second query param: ' + this.secondQueryParam);  
}
```

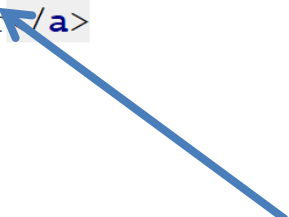


home works!

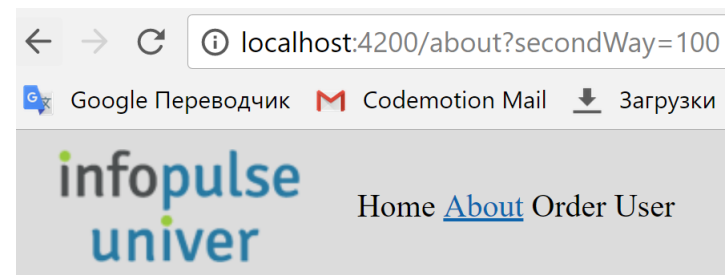
Передача query через routerLink

Конечно же существует возможность передать **query** параметры не только через наш код, но и через **html**.

```
<div class="navigation-bar">
  <a [routerLink]="['']">Home</a>
  <a [routerLink]="['about']" [queryParams]="{'secondWay': 100}">About</a>
  <a [routerLink]="['order', orderId]">Order</a>
</div>
```



Обратите внимание, что **property binding** [queryParams], работает только в паре с **директивой** [routerLink]



about works!

Navigate to home

Child routes

Почти каждом приложении возникает необходимость создавать иерархию роутинга.

Представьте, что у вас есть раздел «**Пользователи**». По ранее изученной схеме файл маршрутизации выглядел бы так:

path: 'user/:id', component: UserComponent

path: 'user/:id/details', component: UserDetailsComponent

path: 'user/:id/orders', component: UserOrdersComponent

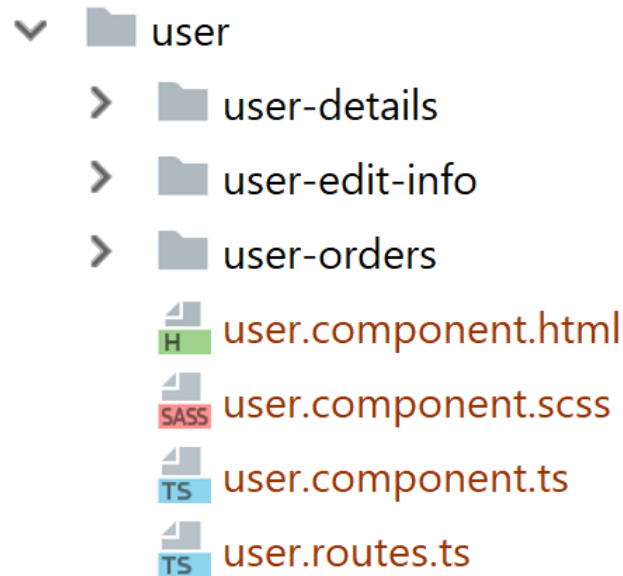
path: ' user/:id/editInfo', component: UserEditInfoComponent

Странно что приходится дублировать путь **user**, не так ли?

Именно для таких ситуаций используют **Child Routes**. На примере видно что **UserDetailsComponent, UserOrdersComponent, UserEditInfoComponent**, являются дочерними по отношению к компоненту **UserComponent**.

Child routes

Модифицируем структуру проекта – добавим еще четыре компонента, один главный и 3 вложенных



Далее необходимо создать файл маршрутизации, для дочерних компонентов, компонента **User**.

Создадим файл **user.routes.ts** в корне папки **users**

Child routes

Содержимое файла `user.routes.ts`

```
import {Routes} from '@angular/router';
import {UserDetailsComponent} from '../user-details/user-details.component';
import {UserEditInfoComponent} from '../user-edit-info/user-edit-info.component';
import {UserOrdersComponent} from '../user-orders/user-orders.component';

export const USER_CHILD_ROUTES: Routes = [
  {path: 'user-detail', component: UserDetailsComponent},
  {path: 'user-edit-info', component: UserEditInfoComponent},
  {path: 'user-orders', component: UserOrdersComponent}
];
```

Внесем изменения в главный файл маршрутизации `app.routing.ts`

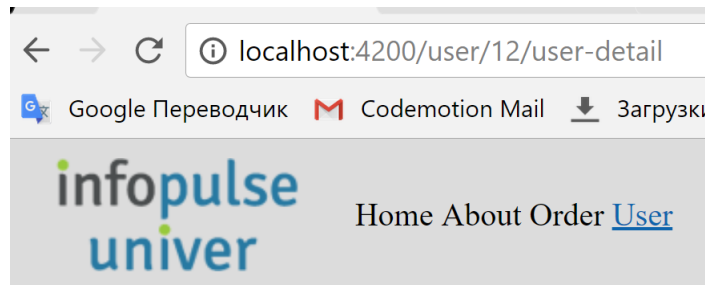
```
const routes: Routes = [
  {path: '', component: HomeComponent},
  {path: 'about', component: AboutComponent},
  {path: 'order-data/:id', component: OrderDataComponent},
  {path: 'user/:id', component: UserComponent, children: USER_CHILD_ROUTES},
];
```

Child routes

По сколько мы «сказали» **Angular**, что будем использовать **child routes**, то необходимо указать месторасположение вложенного **<router-outlet>** в котором и будет отображаться дочерний контент.

```
<div class="sub-navbar">
  <a [routerLink]="['user-detail']" routerLinkActive="active-user-link">User Detail || </a>
  <a [routerLink]="['user-edit-info']" routerLinkActive="active-user-link">Edit user info || </a>
  <a [routerLink]="['user-orders']" routerLinkActive="active-user-link"> User orders ||</a>
</div>
<div class="user-content">
  <router-outlet></router-outlet>
</div>
```

Таким образом, в приложении в данный момент должно быть **два <router-outlet>**. Один для отображения контента основных маршрутов, а **второй** для отображения внутреннего контента users.



User Detail || Edit user info || User orders ||

user-details works!

Редирект

Так же, **Angular** предоставляет возможность создания **редиректов**.

Давайте посмотрим ближе на наш конфигурационный файл маршрутизации.

```
const routes: Routes = [  
  {path: '', component: HomeComponent},  
  {path: 'about', component: AboutComponent},  
  {path: 'order-data/:id', component: OrderDataComponent},  
  {path: 'user/:id', component: UserComponent, children: USER_CHILD_ROUTES},  
];
```

Согласно этим настройкам, у юзера нет возможности просто перейти на раздел **Order**, по скольку в настройках указан маршрут **order/id**.

Используя директиву **redirectTo**, мы можем перенаправить пользователя на любую другую страницу. Например на первый заказ пользователя.

```
{path: 'order-data/:id', component: OrderDataComponent},  
{path: 'order-data', redirectTo: 'order-data/1'},
```

Задание

- Реорганизовать структуру проекта и роутинга с использованием Child Routes
- Создать страницу 404 и перенаправлять любой не существующий запрос на эту страницу
- По возможности и необходимости – использовать работу с параметрами