ROUTING

JOHAN KUSTERMANS

PLANNING

- Basics
- Guards & Resolvers
- Auxiliary Routes
- Events
- Lazy Loading

BASICS

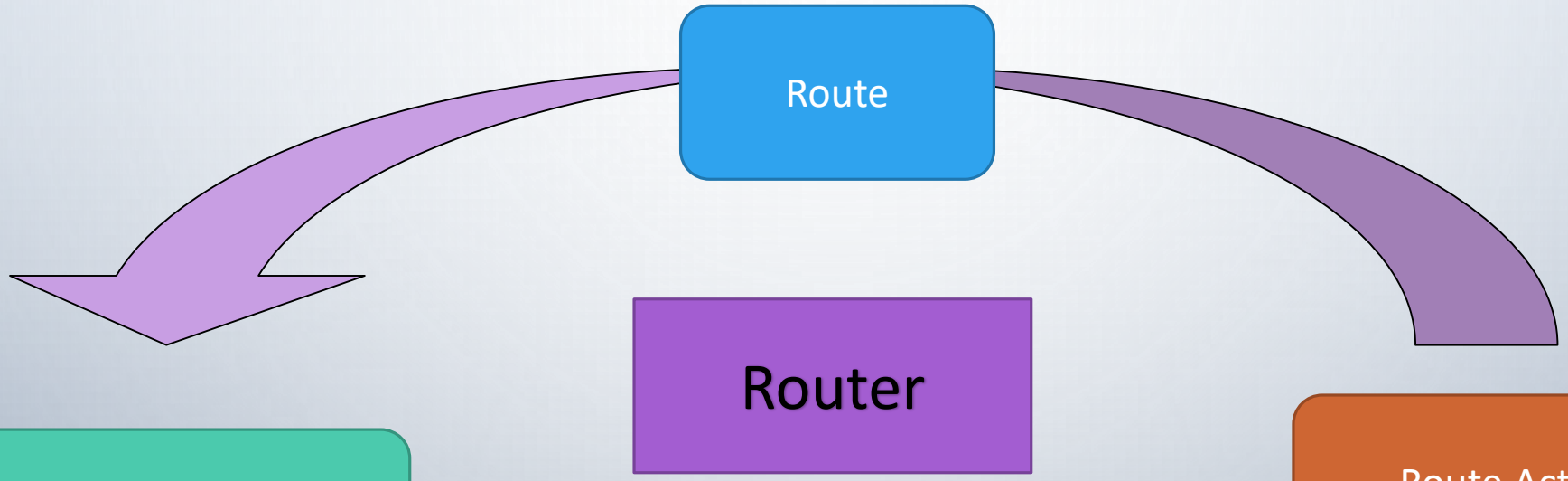
What is Routing?

Routing is the API that describes navigation in an Angular application

- Navigation between screens
- Navigation to sub components
- Showing popups besides components (more advanced)

ROUTING

```
{ path: "contacts", component: ContactScreen }
```




```
…
<router-outlet>
…
```

target_component.html

```
…
<a routerLink="/contacts" >
…
```

Exercise

1. In “application.module.ts”, add 2 routes to applicationRoutes:
   - A route with path ‘contacts’, pointing to the ContactScreen component
   - A route with path ‘inbox’, pointing to the InboxScreen component.

3 Add the router links to ApplicationHeader so that we can navigate to the 2 screens mentioned above.

4 Add a <router-outlet></router-outlet> to your application.

ROUTING

- Imperatie (through code)

```
router.navigate(["contacts"]);
```

- URL Link

http://localhost:9001/contacts/contact/2

PARAMETRIZED ROUTING

- Parameters

```
{ path: "contact/:id", component: ContactEditor }
```

```
<a routerLink="['contact',2]" >
```

http://localhost/contact/2

```
<a routerLink="['contact',2,{name:'Donald',age:71}]" >
```

http://localhost/contact/2;name=Donald;age=71

- Child routes

```
{path: "contacts", component: ContactScreen,

        children : [
                { path: "contact/:id", component: ContactEditor }
        ]

}
```

Router outlet

Parametrized Routing / Child Routes

- Parameters: scoped to route

```
{ path: "contact/:id", component: ContactEditor }
```

```
<a routerLink="['contact',2]" >
```

http://localhost/contact/2

```
<a routerLink="['contact',2,{name:'Donald',age:71}]" >
```

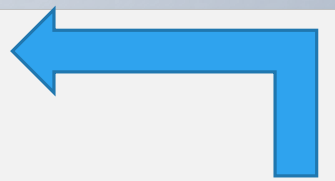
http://localhost/contact/2;name=Donald;age=71

- Child routes

```
{path: "contacts", component: ContactScreen,

        children : [
                { path: "contact/:id", component: ContactEditor }
        ]

}
```



Router outlet

Parametrized Routing

- Local Router State : via ActivatedRoute

  - ActivatedRoute.snapshot

    The params & queryParams are just objects (key-value's)

```
route.snapshot.params["id"]
route.snapshots.queryParams["a"]
```

  - ActivatedRoute

    The params & queryParams are observables

```
route.params.subscribe(pars => … use pars["id"] … )
```

    See also the pluck operator for observables.

Exercise

1. In “application.module.ts”, add a child route to the ‘contacts’ route:
   - A ‘contact’ route, pointing to the ContactEditor component. This route will be used to show the contact that is selected (in the list) in the editor. What does this mean for the path?
2. For each <a> in the contact-list-viewer-with-routing component, add a router link for the route defined above.
3. Add another <router-outlet></router-outlet> so that it can be filled with an editor. Where should you put it?
4. In the close-method of the ContactEditorWithRouting component, navigate back using the router.navigate method.

Query Parameters

- Query Parameters: not scoped to route

```
{ path: "contact/:id", component: Cont
```

http://localhost/contact/2?name=Donald&age=71

```
<a routerLink="['contact',2]" [queryParams]="{name:'Donald',age:71}">
```

```
<a routerLink="['contact',2]" preserveQueryParams>
```

http://localhost/contact/2?a=1&b=1

In this case "?a=1&b=1" were the query parameters before we navigated via this link

- Imperative

```
router.navigate(['contact',2],{qu
```

```
router.navigate(['contact',2],{preserveQueryParams:true})
```

GUARDS & RESOLVERS

Guards

- Scenario:
  - The user selects a contact in the list and starts editing
  - Then he selects another contact or selects another link in the navigation bar
  - User friendly behaviour: check if he has changed something in the editor and ask a confirmation to lose changes or cancel the navigation (or ask to save the changes).
- Solution: CanDeactivate guard

```
interface CanDeactivate <C> {

  canDeactivate(comp:C, route:ActivatedRouteSnapshot, state:RouterStateSnaphshot) :

                Observable<boolean>|Promise<boolean> {
                        ……
  }

}
```

Guards

```
{ path: ”contact/:id”, component: ContactEditor,
                                    canDeactivate: [EditCancelationGuard]
}
```

Data Resolvers

- Scenario:
  - The user selects a contact in the list.
  - In order to edit the contact we need to fetch details.
  - We only want to show the editor when the details have arrived. If something goes wrong, we want navigation to be canceled.
- Solution: Resolve

```
interface Resolve <C> {

  resolve(route:ActivatedRouteSnapshot, state:RouterStateSnaphshot) :

              Observable<any>|Promise<any>|any {

                    … fetch date and return it …
  }

}
```

Data Resolvers

```
{ path: ”contact/:id”, component: ContactEditor,
        resolve: {
                contact: ContactEditorResolver
        }
}
```

```
class ContactEditor implements OnInit {

   constructor(private route:ActivatedRoute) {}

   ngOnInit() {
       this.contact = route.snapshot.data[“contact”];
   }
```

AUXILIARY ROUTING

EVENTS

LAZY LOADING

ROUTING: REFERENCES

- angular.io / docs
  - Tutorial / Routing
  - ADVANCED / Routing & Navigation

- Victor Savkin (main contributor to Angular, creator of Routing Module but it is “inspired” by ngrx/router)
  - Blog
  - Book: Angular 2 Router

- Blogs Thoughtram
  - http://blog.thoughtram.io/angular/2016/06/14/routing-in-angular-2-revisited.html